Finding Lagrangian structures via the application of braid theory

Michael R. Allshouse

October 16, 2010

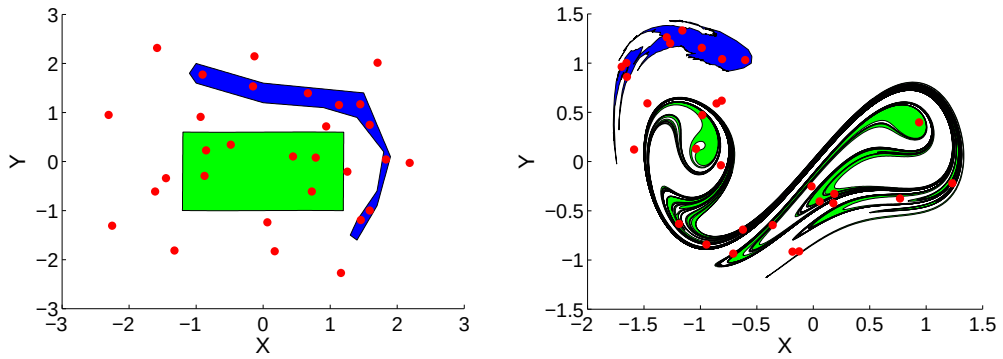
1 Introduction

The ability to accurately identify regions of mixing in two dimensional systems has applications in ocean [1] and geophysical systems [5], as well as granular flows [3]. Over the last decade, much work has been put into finding coherent structures in the Lagrangian frame ([7], [8], and [9]). Most of these methods require field data (e.g. velocity or vorticity fields) which in situations like the ocean are not readily available. However, trajectories of tracers for these systems can accurately be measured and are available for analysis. This indicates that a method which estimates the location of Lagrangian structures based on trajectory data would be highly valuable.

The Lagrangian structures we are looking for are defined as regions of mixing surrounded by a transport boundary. These moving regions contain fluid which mixes with other fluid within the boundary but not with fluid outside the region. This self mixing region can move through the full system and can change shape. The transport boundaries in the fluid effectively divide up the fluid into regions of qualitatively different dynamics. What separates these transport boundaries from other material lines in the fluid is that the length of these boundaries stays the same order of magnitude throughout the time evolution. If the material line perimeter stays of the same order of length then there is negligible mixing occurring between fluid inside and outside of the transport boundary.

Throughout this report, a sample system will be used to demonstrate many of the features and concepts presented. The application of this system to two regions of fluid is presented in Section 3.3. Figure 1a depicts thirty trajectories as well as two regions of interest (blue and green) in an initial state. At a later time the position of the trajectories and the regions of interest are presented in Figure 1b.

While the boundaries of the two regions distorts the trajectories that start within a region remain in that region for all time. An important observation is that while the blue



(a) (b)

Figure 1: (a) Initial state (b) State after application of system described in Section 3.3. The blue region is surrounded by a mixing transport boundary. The green region is not. Float positions are shown as red dots.

region has changed shape, the perimeter has remained approximately the same length. This indicates that the boundary to the blue region would be classified as a mixing transport boundary. The green region shows substantial folding and stretching of the boundary as trajectories from outside the region try to mix with points inside the region. Thus it can be concluded that, this boundary would not be classified as a mixing transport boundary.

The goal of this project is to develop a method for finding Lagrangian structures, like the boundary of the blue region in Figure 1, from trajectory data. We will use the property of a negligibly growing perimeter as the differentiating characteristic between transport boundaries and other material lines in the fluid. In order to do this, we have developed an algorithm that analyzes the trajectory data and makes predictions of where these structures are located. The novelty of this algorithm is its reliance on the tools of braid theory.

The concept of using braid theory for the analysis of trajectories was presented by Thieffeault [11] in the context of quantifying Lyapunov exponents in mixing two dimensional fluids. He applies tools from braid theory to calculate the topographical entropy of a system which is a measure of the mixing occurring in the system. The present research takes this work as a basis but develops some of the concepts in a direction which finds the mixing transport boundaries in the system.

This report is structured such that a number of definitions and an introduction of the necessary background of braid theory is first presented. This will be followed by a demonstration that the tools from braid theory can find mixing structures. Next, a revised algorithm will be presented which will make a braid theory based method a reasonable

option for analyzing large numbers of trajectories in search of transport boundaries. Finally, an application of this method to a mixing system is presented with concluding remarks and future works.

2 Braid theory

In order to fully understand the algorithms presented in this report, a number of definitions and tools from braid theory will be provided here. These definitions will be given in the context of applying this theory to a set of trajectories evolving in time on a two dimensional plane. An example set of trajectories for three floats are shown in Figure 2a to provide the reader with a visual example to follow.

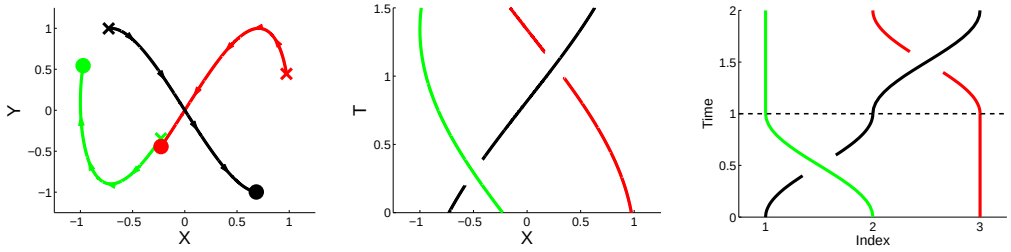
The trajectories are constrained to two dimensions in the physical space, and the position of a float at any given time will be referred to as a *puncture*. For example, there are three punctures in Figure 2a and are represented by colored dots. The initial positions of the punctures are also presented in the figure as crosses.

The trajectories that the punctures traverse can be projected into a three dimensional space where the horizontal plane is the physical domain and the vertical axis is time. These three dimensional trajectories are called *strands*. Due to the monotonic nature of time, a strand can only travel upwards because it can not travel backwards in time. If these three dimensional strands are projected onto the plane containing the x-axis and time then Figure 2a becomes Figure 2b.

The collection of strands make up what is called the *braid*. The braid is uniquely defined by how the strands intertwine, or cross in the projected view. The strands itself can be perturbed and the braid will be considered unchanged as long as the perturbations do not change the direction, number or order of the crossings. The braids shown in Figure 2b and 2c are the same braid even though the strands have been transformed to a different location and over a different time. They reason the two images depict the same braid is because the green strand passes in front of the black strand first then the black in front of the red strand in both cases.

2.1 Generators

Given the importance of the crossing of strands in the braid, it is necessary to define what in braid theory are called *generators*, σ. Generators act on a braid by taking a pair of strands and crossing them. For example the braid shown in Figure 2 has two generators acting in a sequence. The first generator causes the green strand to pass in front of the black strand, or equivalently causes the strand in the first index location to pass in front of the strand in the second index location. The second generator causes the black or second index strand to



(a) (b) (c)

Figure 2: (a) Three sample trajectories shown in the physical plane. An "x" represents the initial condition of the trajectory and a dot represents the current position. (b) The projection of the trajectories onto the x-axis as a function of time. When a trajectory passes behind another trajectory a gap is placed in the trajectory. (c) The transformation of the braid in (b) to a plot of their indexes as a function of time.

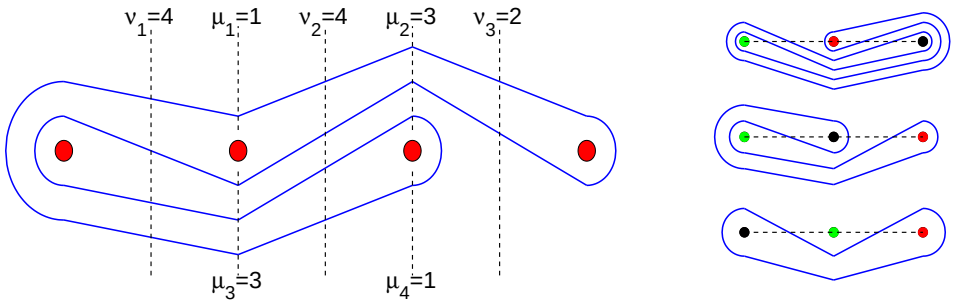
pass in front of the red or third index strand. Note that the black strand has moved from the first to the second to the third index during the application of these generators.

Based on the description of what the generator does it is clear that there are two defining characteristics: first, what strands are involved in the crossing and second, which strand passes in front. These two components can be used to give a numerical value for a specific generator. The strands involved in the crossing are indicated by the magnitude of the generator, and the direction of crossing is indicated by the generator being positive or negative. Since the two strands which cross must be right next to each other, the magnitude of the generator is equal to the lower index of the strands involved in the crossing.

For example, the braid shown in Figure 2c has two generators acting on three strands. The first generator crosses the lower index strand, black, behind the higher index strand. This direction of crossing is defined as positive. The two strands involved in the first crossing are in positions one and two. This indicates that the first generator is $\sigma_1 = 1$. The same procedure can be repeated for the second crossing which takes the strand in index two and crosses it in front of the strand in index three. Thus, the second generator is $\sigma_2 = -2$. In this case, the generator sequence $\sigma = [1 \quad -2]$ fully defines the three strand braid. It should be stressed that the actual strand involved in a crossing does not mater just the indexes where crossing is taking place.

2.2 Loops

The next tool which will be used throughout this report is the concept of a *loop*. A sample loop is drawn in Figure 3a. A loop is a boundary of a region that can pass around or encapsulate punctures and provides a way for us to divide up the fluid system into regions.



(a) (b)

Figure 3: (a.) Sample loop drawn for a four puncture system. The μ's and ν's are presented in the plot and the corresponding Dynnikov coordinates are $a = [1 \quad -1]$ and $b = [0 \;\; 1]$. (b.) How an initial loop (shown at the bottom) gets modified by the generators for the system shown in Figure 2. The number of intersection (from bottom to top) are 2,3, and 5.

If a mixing transport boundary exists and contains punctures, it should be possible to find a loop that represents this boundary. A loop representing a mixing transport boundary must have a length that does not grow with time. Clearly if a loop represents the transport boundary it is important to understand some characteristics of loops.

The first important concept is that a loop can be uniquely defined by a coordinate system. The coordinate system uses information on the number of times the loop passes above, below, or between punctures. For example, the number of times the loop passes directly above the second puncture is represented as μ_1 and below the second puncture is μ_2. Similarly the number of times the loop passes above and below the third puncture are μ_3 and μ_4 respectively. Additionally the number of times the loop passes the midpoint between punctures is also required for the coordinate system representation. To visualize this, a vertical line at the midpoint between punctures is drawn in Figure 3a and the intersection with the loop can be counted. The three midpoint crossings from left to right are ν_1, ν_2, and ν_3.

The information about number of passings can be condensed into a minimal coordinate system called the Dynnikov coordinates [4]. Taking the information about number of passes, the Dynnikov coordinates transform the μ's and ν's by the following relationship

$$a_i = \frac{\mu_{2i} - \mu_{2i-1}}{2} \qquad b_i = \frac{\nu_i - \nu_{i+1}}{2}$$

These coordinates are essentially a comparison of the number of times a loop passes above or below the $i+1$ puncture, a_i, and the difference in the number of times the loop passes to left and to the right of the $i+1$ puncture, b_i. The Dynnikov coordinate representation

for a given loop is $\mathbf{u} = (a_1, ..., a_{n-2}, b_1, ..., b_{n-1})$. This coordinate system uniquely defines any loop by a sequence of integers, and any even number of integers can be represented as a loop.

With the coordinate system in place it is possible to concisely represent any loop and the next step is to understand how the loop evolves as the generators cause the punctures to change positions. As the generators are applied to the strands, the loop surrounding the strands is also moved. An example application of generators to a loop is shown in Figure 3b. The initial loop shown at the bottom of the figure is first modified by the generator $\sigma_1 = 1$ then by $\sigma_2 = -2$ in the same way the strands from Figure 2c are intertwined.

One of the strengths of braid theory is that there is a set of rules for updating the coordinates presented in [10]. These rules allow for the rapid calculates of how the loops changed position due to a generator and prevent the need for a time intensive advection calculation which would require a velocity field. The simple set of update rules also allows for the analysis of a large number of loops simultaneously which aids in the search for the mixing transport boundaries.

Since we have defined mixing transport boundaries as boundaries with approximately time independent length, we need to establish a way to measure loop lengths. Moussafir demonstrated that the length of a loop is proportional to the number of times the loop crosses a line connecting all the punctures. Figure 3b depicts the line connecting the punctures, and an intersection occurs every time the loop crosses this line.

Fortunately, an intersection calculation exists which produces a value proportional to the loop length. The intersection for a given loop, $L(\mathbf{u})$ [10], is calculated by

$$L(\mathbf{u}) = |a_1| + |a_{n-2}| + \sum_{i=1}^{n-3} |a_{i+1} - a_i| + \sum_{i=0}^{n-1} |b_i| \tag{1}$$

where

$$b_0 = -\max_{i \le i \le n-2} \left(|a_i| + max(b_i, 0) + \sum_{j=1}^{i-1} b_j \right) \qquad \text{and} \qquad b_{n-1} = -b_0 - \sum_{i=1}^{n-2} b_i$$

This calculation will be used to determine if a loop is growing with time, and thus not a mixing transport boundary, or if the length stays of the same approximately the same length throughout.

Finally, the set of punctures inside by a non-growing loop will be referred to as a *structure* or puncture structure set. A puncture is considered to be inside a loop if there is no set of generators that will result in the loop not being both above or below the puncture. The punctures (trajectories) inside the loop (transport boundary) will intermix but not mix with

the punctures outside of the loop.

2.3 Summary of the application of braid theory

With the tools and definitions developed in the previous sections, it is now possible to give an overview of the application of braid theory to the problem of finding two dimensional mixing transport boundaries. The trajectories are converted to strands and a generator sequence is calculated from the trajectory crossings. A mixing transport boundary is the equivalent to a loop, which does not show long term growth, surrounding a subset of punctures. The goal of this report is to demonstrate that these loops do exist and can be found.

The search for structures and non-growing loops will be done in two ways. The first method is a methodical search for the non-growing loops by testing Dynnikov coordinates in a sequential manner which is presented in Section 3. This method will be improved upon by a refined searching technique described in Section 4. For both search methods, a system, see Section 3.3, is analyzed containing a mixing transport boundary.

One assumption made throughout this paper, is that the mixing transport boundaries simply surround a region of the fluid. It seems unlikely that a boundary which weaves in and out of other punctures would not grow with time. Based on this, it is assumed that the Dynnikov coordinates defining the non-growing loops will have small magnitudes for its integer coordinates, because as the coordinate magnitudes grow so does the complexity of the loop.

3 Proof of concept

The basis of the proof of concept method is that if a simple non-growing loop, or mixing transport boundary, exists it will be found by analyzing the set of all simple loops. Here we present what is classified as a simple loop, an outline of the code which calculates and analyzes the loops, and a sample system and results of the proof of concept code.

3.1 Simple loops

The Dynnikov coordinates provide a compact way of representing loops, and any list of $2n-4$ integers can create a loop for an n puncture system. Additionally, the loop grows more and more complex as the coordinate values grow larger. Based on the assumption that the non-growing loops are simple loops, the proof of concept code will only search through loops where each of the initial coordinates can have a value between -1 and 1. Four example loops are presented in Figure 4 demonstrating the range in complexity which is encompassed by this range of Dynnikov coordinates.

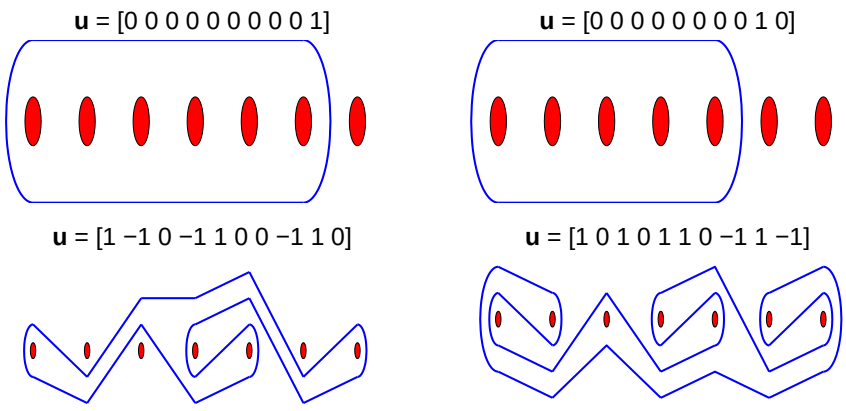


Figure 4: Four loops which fall into the category of simple.

3.2 Algorithm outline

In order to find the non-growing loops, an algorithm is developed using Matlab. The objective of this algorithm is to take in a set of trajectories and output the non-growing loop entangling the punctures. The first step is to calculate the generator sequence which defines the braid of the trajectories. Next, for each Dynnikov coordinates which has coordinate values between -1 and 1 inclusively, the entire generator sequence is applied and the resulting loops have their intersections calculated. The loops with the smallest final number of intersections are designated as non-growing loops.

3.2.1 Calculation of Generators

The generator sequence for a given set of trajectories is calculated by a procedure called the color braiding method. Pairs of strands are analyzed to determine when the strands cross. This method is called "colored" because each strand is distinguishable throughout searching through the pair and it matters which strand starts in the lower index position. This differs from the final generator sequence where only the indexes of the crossing are considered and not which strands are involved in the crossing.

The color braiding search starts first by ordering the braids based on the initial value of their $x-$coordinate. Then the code loops through each pair of strands where there are $(n^2 - n)/2$ pairs for systems with n strands. For the pair of strands, three pieces of information are saved for each crossing: the time of crossing, the direction (either clockwise or counter-clockwise based on the y value at the time of crossing), and the order of the strands before crossing. After all pairs of strands have been analyzed, the crossings are collected and ordered by time of crossing.

The final step is to convert the list of crossings to a generator sequence. In the cur-

$t_{crossing}$	Direction	Left Strand	Right Strand
1.245	1	2	3
2.142	1	1	3
3.145	-1	1	2

$$
\begin{array}{lcllcl}
I & = & [1\,2\,3], & & & \\
I & = & [1\,3\,2], & \sigma_1 & = & 2 \\
I & = & [3\,1\,2], & \sigma_2 & = & 1 \\
I & = & [3\,2\,1], & \sigma_3 & = & -2
\end{array}
$$

Figure 5: A sample system featuring three strands and three crossings. The left table demonstrates the information from the comparison between pairs of strands. The right side shows the index vector as its being changed by the crossings and the resulting generator sequence.

rent state, information on which strands cross is available but the generator sequence is dependent on the index positions involved in crossings. This conversion is accomplished by creating a vector naming each strand after the strands initial index and applying each crossing chronologically. For each crossing, the two strand names are found and their positions in the vector are switched. The smaller index of the strand locations becomes the generator magnitude and the direction determined by color braiding is applied to calculate the generator. The calculated generator sequence is then saved with the corresponding times. A sample system is shown in Figure 5 and outlines the conversion from an ordered list of color-braiding information to a generator sequence..

3.2.2 Calculating loops, applying generators, and intersection calculation

The next step in the algorithm is to cycle through all possible simple loops and apply the generator sequence. As mentioned in Section 2.2, an n puncture system is represented by $2n-4$ coordinates and only values of -1 ,0, and 1 will be considered for each coordinate. This results in $3\hat{}(2n-4)-1$ possible loops which must be checked (Note: the loop with all coordinates equal to zero is not considered a loop).

A simple methodical method for representing all these loops is to create a list from 1 to $3\hat{}(2n-4)$ and convert the list to their respective values in base 3. The resulting numbers are then converted to a string of length $2n-4$ where the necessary number of zeros have been appended to the left side of the string. Finally, each value in the string is reduced by one producing the Dynnikov coordinates. With each loop the generator sequence is applied by utilizing the transformation rules and a final loop is ready to be measured.

The last step of the algorithm is to take each final loop and calculate the number of intersections. This is a simple calculation based on Equation 1. Given that the number of intersections is proportional to its length, a criterion is set by the user which will define a loop as growing or not growing. If the number of intersections falls below this threshold it represents an approximation for the location of a mixing transport boundary in the fluid.

3.3 Modified Duffing Oscillator system

In order to test this algorithm, it is necessary to develop a system where there is a mixing transport boundary. This system's velocity field will be used to calculate trajectories for analysis. While this would not be the target application of the algorithm, having a test case where it is possible to determine *a priori* which punctures make up a structure.

The system selected for this test is a modification of the Duffing Oscillator. One change is made in order to create two mixing regions and the other change adds solid body rotation. The base system without rotation, $\mathbf{x} = f(\mathbf{x})$ is defined as

$$\begin{aligned} \dot{x} &= y + \alpha\cos(\omega t) \\ \dot{y} &= x(1-x^2) + \gamma\cos(\omega t) - \delta y. \end{aligned}$$

where $\alpha = .1$, $\gamma = .14$, $\delta = .08$, and $\omega = 1$. To add rotation to the system, the following transformation is made

$$\begin{aligned} \tilde{\mathbf{x}} &= [R(t)]\mathbf{x} \\ \dot{\tilde{\mathbf{x}}} &= \left[\frac{d}{dt}R(t)\right]\mathbf{x} + [R(t)]\dot{\mathbf{x}} \\ \dot{\tilde{\mathbf{x}}} &= \left[\frac{d}{dt}R(t)\right][R(t)]^{-1}\tilde{\mathbf{x}} + [R(t)]f\left([R(t)]^{-1}\tilde{\tilde{\mathbf{x}}}\right) \end{aligned}$$

where

$$[R(t)] = \begin{bmatrix} \cos(\Omega t) & \sin(\Omega t) \\ -\sin(\Omega t) & \cos(\Omega t) \end{bmatrix}.$$

The key feature of this system is that there are two mixing regions. With the velocity field, it is possible to find these regions at the initial time, and a plot of the two regions is presented in Figure 6. This plot indicates that all points initially in the gray region should make up a structure, and all punctures initially in the white region should make up a second structure. Each of these structures should have a non-growing loop surrounding the punctures in the given structure.

3.4 Results of basic algorithm

To test the proof of concept algorithm a variety of sets of initial conditions are tested. A single representative example is presented here. This system consists of seven initial conditions which are shown in Figure 6. Based on that figure, it is expected that four of the punctures should make up one structure and the other three punctures will form a second

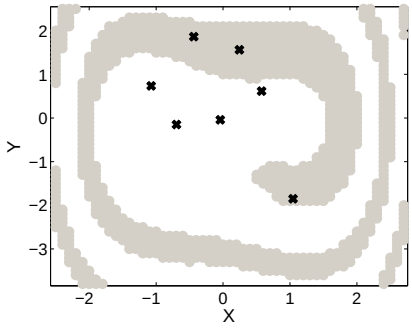
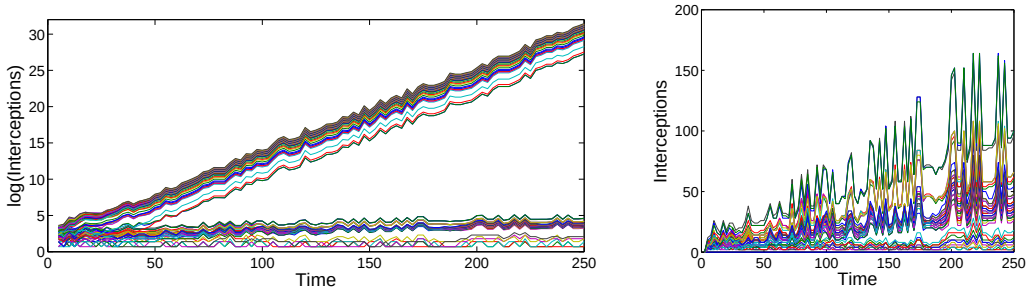


Figure 6: Mixing regions for the modified Duffing oscillator system. The white region collapses to a chaotic mixing region at the center. The gray region approaches a limit cycle around the white region. The black x's are the initial conditions for the trajectories discussed in Section 3.4.



(a) (b)

Figure 7: (a) Semi-log plot of the intersections as a function of time for all possible loops with Dynnikov coordinate values ranging from -1 to 1. (b) Plot of the intersections as a function of time for loops where growth appears to be non-exponential.

structure. With these trajectories, the corresponding generators are calculated and the 3^{10} loops are manipulated via the transformation rules. The lengths of the resulting loops are then calculated.

For a representative set of loops in the defined range, the log of the intersections as a function of time are plotted in Figure 7a. This plot indicates that there are two regimes of growth: loops growing exponentially (linearly increasing lines in the plot) and loops that show non-exponential growth (lines approximately flat in the plot). The loops growing exponentially will not correspond to loops surrounding a structure, so they can be neglected. It should be noted though that they all share the quality that the loops enclose a subset of the four punctures which mix chaotically in the center but never all four.

A closer view of the non-exponentially growing intersections are shown in 7b. This plot is not on a log scale and shows that some of the loops grow approximately linearly while

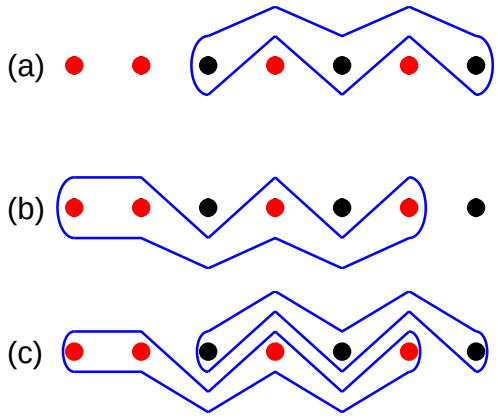


Figure 8: Simplified loop diagrams of the loops which do not show steady state growth. (a) Loop containing the orbiting trajectories (black punctures). (b) Loop containing the inner trajectories (red punctures). (c) Double loop featuring both structure loops.

others don't grow at all. The loops which grow linearly contain all four trajectories in the chaotic region and a subset of the limit cycle region trajectories. The loops which don't show growth, presented in Figure 8, correspond to mixing transport boundaries. Changes in the loop do occur throughout the mixing due to the moving of the particles, but there is no steady state growth of the initial loop.

The expected transport boundaries were found indicating that the proof of concept successfully applied the tools from braid theory to the two dimensional trajectories. One drawback from the proof of concept method is that it is limited in application due to the run times. For systems of eight or fewer punctures, it takes seconds to find the non-growing loops, but for each trajectory added after eight, the run time increases by almost an order of magnitude. This is due to the bottleneck of the code being the application of generators to loops. This process is repeated for each loop and the number of loops searched is $3\hat{}(2n-4) \sim 9^n$. This fact renders the method useless for systems with more than eleven punctures.

4 Refined loop finding algorithm

While the methodical search through all simple loops produces the non-growing loops, it is not a realistic options for systems where there are thirteen or more punctures (run times would be on the order of four days). The target application for this method is systems with a large number of tracers which mix well so having a capacity for analysis of only thirteen particles is not sufficient. For braid theory to be a reasonable method for finding mixing transport boundaries, it is necessary to develop a code where the time to find non-growing

loops is not $t \sim O(9^n)$.

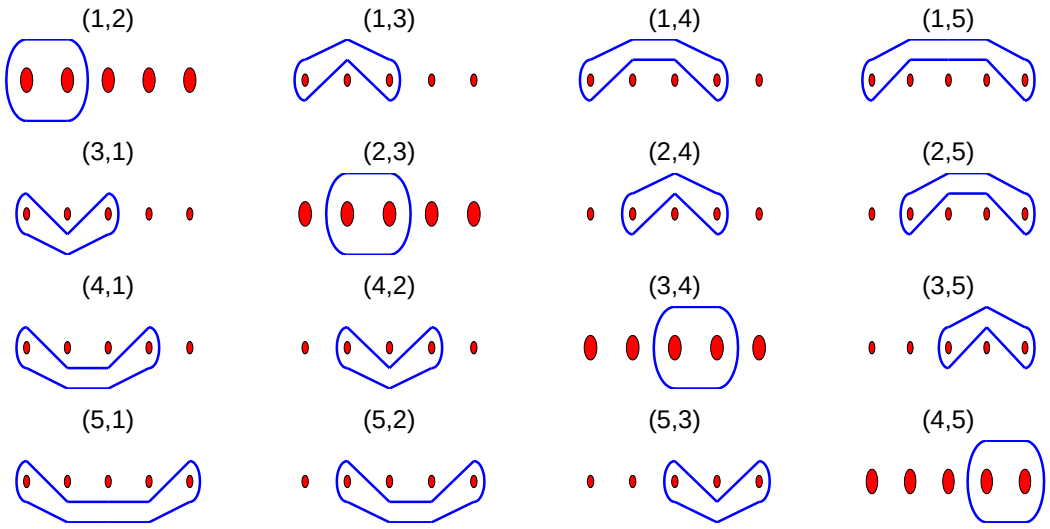
One major drawback of the previous method is that no information is gained from a loop which is growing, so every loop has to be tried. The refined algorithm tries a specific set of loops and analyzes their state after the generators have been applied. The loops analyzed connect two punctures. By letting these loops evolve their final state gives an indication of which punctures become entangled by the loop. Now links can be drawn between which punctures entangle which other punctures. This is the basis for the refined algorithm.

The objective of the refined algorithm is still to take in a set of trajectories and output the punctures within a structure and the non-growing loop which surrounds the loop. The outline for this algorithm is still to first calculate the generator sequence which defines the braid. This generator sequence will be applied to a set of Dynnikov coordinates which connect pairs of punctures, described in Section 4.1. The resulting loops are then analyzed to determine which punctures are entangled by the loop, Section 4.2. With the information on entangled sets, a code determines which punctures form a structure, Section 4.3. Finally, the punctures which make up a structure are surrounded by a loop which does not grow, Section 4.4. After the new algorithm has been described, its application to the modified Duffing oscillator will be presented.

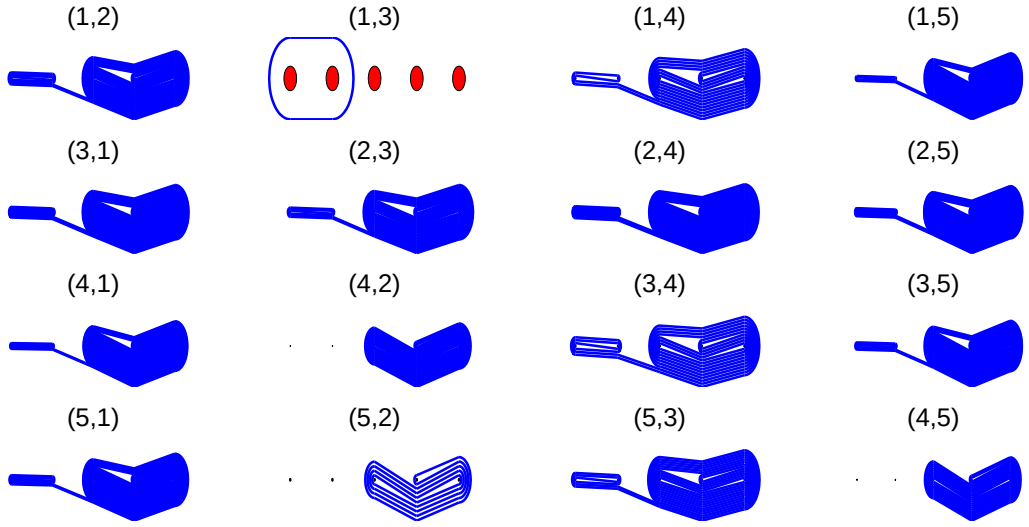
4.1 Pair-loops

As mentioned in the description of the algorithm, the loops of initial interest are ones which connect a pair of punctures. While there are an infinite number of ways for a loop to connect punctures, the main concern here is how simple loops entangle the other punctures. Given this consideration, punctures that neighbor each other will be connected like loop (1,2) in Figure 9a. For punctures that are not neighbors, two loops are created where one loop passes above intermediate punctures, loops (1,3) and (1,4) in Figure 9a, and the other passes below, loop (3,1) or (4,1) in Figure 9a.

For a system of n particles there are $n-1$ loops connecting neighbors and $(n-2)(n-1)$ loops connecting above and below resulting in a total of $(n-1)^2$ loops to analyze. In the last method, the bottle neck was the application of the generators to the $3\hat{}(2n-4)-1$ loops causing the run time to be $t \sim O(9^n)$. Now if the bottle neck of the algorithm is still the application of the generators, then at least the run time has become $t \sim O(n^2)$ which grows substantially slower particularly for large n. Figure 9a is an example for the initial loops which are considered for any five puncture system.



(a)



(b)

Figure 9: (a.) The initial loops for a five puncture system. The notation for each initial loop is (a, b) where the loop goes from a to b in a clockwise direction. (b.) The loops which result from applying a generator sequence. Only loops (1,3), (4,2), (5,2), and (4,5) do not entangle all punctures after generator application. This group of trajectories form two puncture structure sets (1,3) and (2,4,5)

4.2 Calculation of entangled punctures sets

After the generator sequence has been applied to the pair-loops, the code needs to determine which punctures are entangled by the loops. We will describe a puncture as disentangled with a loop if the loop does not pass above and below the puncture. To determine this the final loop Dynnikov coordinates are converted to the PMN-coordinate system. The PMN-coordinate system calculates the number of times the loop passes above (M_i) and below (N_i) the ith puncture as well as half the number of intersections of the loop with the mid-line between punctures ($P_i = \nu_i/2$). For example the loop presented in Figure 3a has $M = [2\ 1\ 3\ 1]$, $N = [2\ 3\ 1\ 1]$, and $P = [2\ 2\ 1]$.

With this new coordinate representation, it is straightforward to determine if a puncture is entangled or not. For the ith puncture to be disentangled either M_i or N_i is zero for the loop after applying the generator sequence. The code searches for all entangled punctures for a given loop and then saves entangled set and the corresponding initial loop. This process is repeated for each pair-loop. After all loops have been processed, the entangled sets are condensed to remove any repeated sets.

For the example presented in Figure 9, the generators have been applied to the loops in Figure 9a and they produce the loops in Figure 9b. Of the sixteen loops only four do not entangle all the punctures. The code only considers entangled sets which are a subset of the full set of punctures. The loop (1,3) entangles only punctures 1 and 3. The loops (4,2), (5,2), and (4,5) entangle only the punctures 2, 4, and 5. This means the example results in two entangled sets which also happen to be the structures for this system. It is possible to find a non-growing loop around punctures 1 and 3 and one around 2,4,and 5.

4.3 Determining punctures in a structure

While the example presented in Sections 4.1 and 4.2 produce the structure sets directly from the entangled puncture sets, this is not always the case. For a pair-loop, the two punctures are either within the same structure or not. If a loop connects points within a structure, the loop can either entangle a subset of punctures within the same structure, all punctures within the structure, or all punctures within the structure as well as punctures from another structure. In the simple example, loops connecting points within a structure always only entangled all the points within the same structure. Alternatively, if a loop connects punctures from different structures they will entangle all the punctures from the two structures and possibly other structures as well. The code attempts to take these entangled sets and decipher which punctures form a structure.

To create the structures, the code runs through the entangled sets and allocates a structure index to the set. The entangled sets are read in from largest to smallest, so the

first and largest set is labeled as structure 1 and then the next entangled set is analyzed. If the second set intersects a structure, then all points in the second set are labeled with the same structure index. However, if the second set is disjoint from the structures, it is given a new structure index. This process is repeated for each of the remaining entangled puncture sets. It is possible for an entangled set to intersect multiple structures in which case the structures as well as the entangled set will all merge into one structure.

At the end of this process, there are two options either a set of disjoint structures have been created or a single structure containing all punctures. If the latter option is produced, the code reruns but instead starts with the next smallest entangled set and repeats the process till it creates multiple structures. Based on the method of finding structures, it is very easy for multiple structures to get combined into one larger structure. To combat this issue the code has the capability to rerun the full algorithm on a structure to determine if it can be broken down into smaller structures. If the code runs through all the possible starting positions, and it can still only find a structure containing all punctures, it concludes that there are no structures present and the absence of non-growing loops.

For clarity, the following is an example where there are three known structures $\{1,2\}, \{3,4\}$ and $\{5,6\}$. The entangled sets are

$$\{1,2,3,4\},$$
$$\{3,4,5,6\},$$
$$\{1,2\},$$
$$\{3,4\},$$
$$\{5,6\}.$$

Starting with the first entangled set leads to the formation of one structure after trying four sets. Since only one structures is found, the code restarts from the second entangled set. This results in two structures $\{1,2\}$ and $\{3,4,5,6\}$. The first structure is one of known structure while the second combines two of the known structures. This fact would be resolved after rerunning the $\{3,4,5,6\}$ trajectories through the full algorithm.

4.4 Calculating Dynnikov coordinates for a structure

The final task for the algorithm is to find a loop for each structure which surrounds the punctures and does not grow. A loop connecting the structure's puncture set can be viewed as the sum of the loops connecting punctures in a structure. An example is presented in Figure 10 where three loops connect the punctures in a structure then are merged to form one loop. The task of this portion of the algorithm is to determine how to connect the punctures in a structure then merge these loops together.

If a loop surrounding a structure does not grow and it is composed of links between punctures, then the links necessarily should not grow due to being mixed by punctures

(a) (b)

Figure 10: Punctures 1,2,4, and 6 are in a structure. (a.) The loops shown here connect the punctures. Blue connects 1 to 2, red connects 2 to 6, and green connects 6 to 4. (b.) The connecting loops can be merged to form a single loop around the punctures.

outside the structure. This means that if all punctures from a structure are removed except for two that are linked then the loop that links the remaining structure-punctures should not grow. From the example in Figure 10, if punctures 4 and 6 were removed, the loop connecting punctures 1 and 2 should not grow in a braid of strands 1,2,3, and 5. The code takes advantage of this property, in order to find the links.

This portion of the code takes in the punctures that are in a structure and outputs the loop surrounding the punctures which does not grow. Using the property previously described as the basis for this method, the code selects a pair of punctures in the structure then calculates the generator sequence for a braid containing only these two punctures and the punctures not in the structure. With the generator sequence simple loops connecting the two punctures of interest are created and the generators are applied. The code calculates the number of intersections and determines if there is a non-growing loop. Once all the punctures are linked together, the loops are merged and the final loop is analyzed to confirm it does not grow. This concludes the algorithms task.

4.5 Application to modified Duffing oscillator

The goal of the refined method is to turn the use of the braid theory into a potential method for finding mixing transport boundaries. The run times from the proof of concept code significantly limit that approaches applicability, but by changing the number of loops searched from $O(n^2)$, there is potential in the refined method. For a direct comparison, a set of five n puncture systems are randomly created and run for both algorithms. The results of these times are presented in Table 1.

The run time results demonstrate the drastic advantages that the refined method offers. While systems with fewer than nine punctures actually complete their runs in a faster time using the proof of concept algorithm, the refined algorithm allows the braid theory approach to be applied for systems with much larger numbers of punctures. While the time growth rate for the proof of concept clearly becomes the predicted $t \sim O(9^n)$, the refined algorithm

# of punctures	5	6	7	8	9	10	11	20
Proof of Concept	0.33	0.46	0.70	5.98	53.20	462.18	3445.39	N/A
Refined	6.71	9.46	11.61	12.34	12.87	15.71	20.48	127.86*

Table 1: Run times for the modified Duffing oscillator using the proof of concept and refined algorithms. * The run with 20 generators had over four times as many generators which contributes to this being higher than the trend would suggest.

does not demonstrate a specific trend in growth time. This is due to the fact that the application of generators is not the bottle neck of the algorithm. The larges amount of time is spent calculating the non-growing loop from the structures.

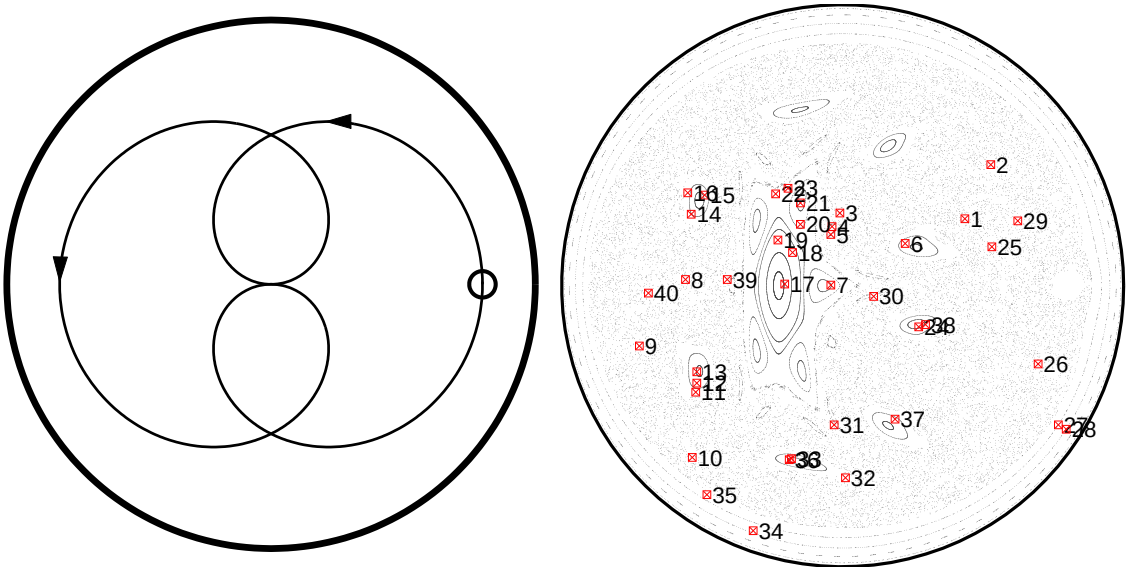
5 Finding non-growing loops in a rod mixed system

The final step in this report is to apply this algorithm to an actual fluid system. The system selected for a first application test is the mixing performed by the translating-rotating mixing system presented by Finn and Cox [6]. Their mixing mechanism features a circular domain of fluid and a circular stirring rod located somewhere in the fluid. The rod has the capability of both translation and rotation about its center and the outer boundary of the domain has the ability to rotate about its axis. The authors are able to develop an analytic solution for the fluid motion when the rod is moved slowly through the fluid.

An interesting result from this type of mixing is the appearance of islands of poor mixing when the circular rod traverses a periodic trajectory through the fluid. These islands have a transport boundary around them which remains the same order of size and mixing inside and outside don't interact. The presence of these islands is undesirable because the purpose of the mixing is to homogenize the entire domain. With the authors analysis it is possible to make accurate estimations on the size and locations of these islands by calculating an iterated or Poincare map. Our objective is to determine if the refined braid theory algorithm can also find these islands.

A sample system is specified by setting the trajectory of the rod, presented in Figure 11a, and a set of initial conditions for the trajectories, presented in Figure 11b. A set of forty initial conditions with half distributed in seven islands and the other half distributed throughout the well mixed chaotic sea. The rods position is also taken as a trajectory bringing the total to forty-one. The initial conditions are tracked for five periods of the mixing, and their trajectories produce a seven thousand generator long sequence.

The application of the refined algorithm takes approximately five and a half minutes and produces the results roughly drawn in Figure 12. The loops plotted are drawn by hand but accurately represent the non-growing loops. An algorithm to represent the non-growing



(a) (b)

Figure 11: (a.) Mixing geometry for sample system. (b.) Iterated map demonstrating the islands and the initial positions of the forty trajectories.

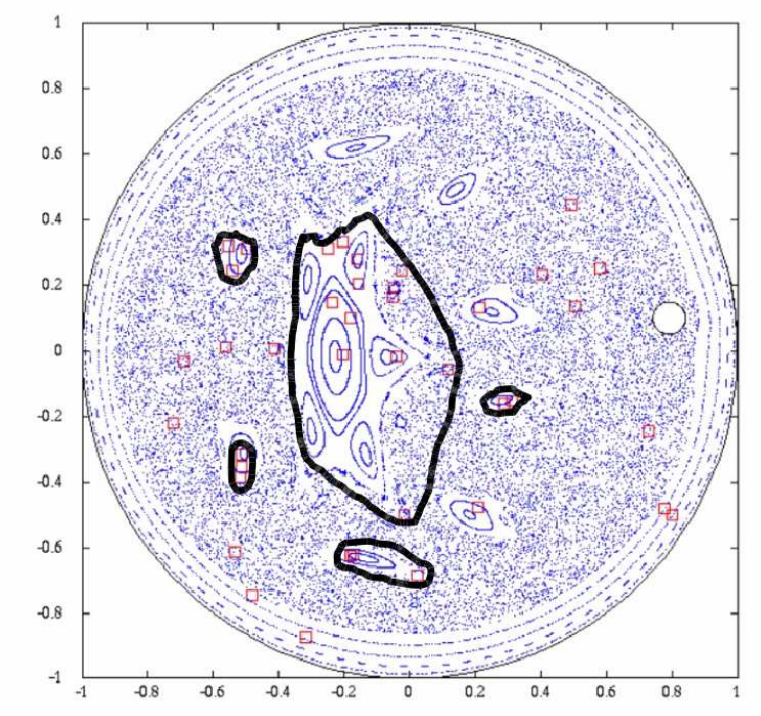


Figure 12: Mixing rod system with the initial conditions and the non-growing loops drawn in.

loops in development.

The first observation is that not all of the islands have a non-growing loop around them. The reason for this is that there are two islands which don't have any trajectories in them and two with only one trajectory. This algorithm relies on multiple trajectories being inside a mixing transport region in order to find them. If there are no trajectories, then the code is unaware that the region exists, and if there is only one trajectory then the code does not recognize a non-growing loop around a single point.

The other observation is that some of the loops include nearby trajectories in the chaotic sea. The largest loop and the loop at the bottom of Figure 12 both are examples of non-growing loops beyond the boundaries of the island. The justification for this is that in the period of time analyzed, none of the punctures in the chaotic sea, the other islands, or the rod pass between the islands and its nearby neighbors which end up in the structure. If more time were analyzed or a larger number of punctures, the probability that a puncture would pass between the island and the nearby puncture becomes higher, and if this passing were to occur the code would find a loop around only punctures in the island.

6 Conclusion

This report has presented the concept of using braid theory to find mixing transport boundaries in two dimensional systems using float data. Introductory information on braid theory is provided for a foundation to the reader, and then a proof of concept algorithm is developed and demonstrated. With the conclusion that the proof of concept algorithm is not applicable to systems with more than thirteen trajectories, a refined algorithm is presented which uses a pair wise method for searching for non-growing loops. This method successfully reduced the computation time and significantly increased the number of trajectories which can be analyzed. This algorithm is then applied to the real fluid system of a viscous fluid being mixed by a moving rod.

Some observations on the limitations of this algorithm are presented in Section 5, but there are more issues than these system specific considerations. In order for the algorithm to work it is necessary that the significant mixing is occurring in the system. The accuracy of the sets of punctures forming structures is directly proportional to the number of generators in the system.

Another potential issue is that the float trajectories have to be highly accurate. As previously stated, it is necessary to have the correct generator sequence in order to find loops which do not grow. If the position data does not have a high resolution, it is possible a crossing will be missed leading to inaccurate structures and false non-growing loops. While it is straightforward to refine a trajectory from a velocity field, this may not be an option

for trajectory data taken from a float in the ocean.

Even though these issues do limit the application of the refined algorithm and braid theory somewhat, there is still the potential to find structures in well mixed systems with a large number of particles. One potential area of application is the search for clustering in granular flow data. There is a need to be able to find clusters of particles which move together and mix amongst themselves [2]. The refined algorithm could be used to take trajectories of grains and find these clusters.

In addition to finding applications of this algorithm there are a number of directions for future work. One improvement on the algorithm would be to add the ability to plot the non-growing loops on the data as the system evolves. Current attempts to plot the non-growing loops has been unsuccessful. It would also be valuable to determine the success rates of the refined code for various crossings per trajectory values. This would give a sharper range of applicability of the refined algorithm.

Acknowledgments

I would like to thank Jean-Luc Thiffeault for introducing me to the world of braid theory, helping with development of the proof of concept algorithm, and general guidance throughout the project. Matt Finn offered help in testing the refined algorithm on the rod stirred system. Additionally, the author would like to thank the rest of the faculty that aided in the Woods Hole summer program as well as the other fellows for an enjoyable and educational summer. Finally, I would like to thank the NSF for providing funding for this wonderful opportunity.

References

[1] F. Beron-Vera, M. J. Olascoaga, and G. J. Goni, *Oceanic mesoscale eddies as revealed by lagrangian coherent structures*, Geophysical Research Letters, 35 (2008).

[2] D. Bonamy, F. Daviaud, L. Laurent, M. Bonetti, and J. P. Bouchaud, *Multiscale clustering in granular surface flows*, Physical Review Letters, 89 (2002).

[3] T. G. Drake, *Structural features in granular flows*, Journal of Geophysical Research, 95 (1990), pp. 8681–8696.

[4] I. A. Dynnikov, *On a yang-baxter map and the dehornoy ordering*, Russian Math Surveys, 57 (2002), pp. 592–594.

[5] C. G. Farnetani and H. Samuel, *Lagrangian structures and stirring in the earth's mantle*, Earth and Planetary Science Letters, 206 (2003), pp. 335–348.

[6] M. D. Finn and S. M. Cox, *Stokes flow in a mixer with changing geometry*, Journal of Engineering Mathematics, 41 (2001), pp. 75–99.

[7] G. Haller, *Lagrangian structures and the rate of strain in a partition of two-dimensional turbulence*, Physics of Fluids, 13 (2001).

[8] G. Haller and G. Yuan, *Lagrangian coherent structures and mixing in two-dimensional turbulence*, Physica D, 147 (2000), pp. 352–370.

[9] M. Mathur, G. Haller, T. Peacock, J. Ruppert-Felsot, and H. L. Swinney, *Uncovering the lagrantian skeleton of turbulence*, Physical Review Letters, 98 (2007).

[10] J.-O. Moussafir, *On computing the entropy of braids*, Functional Analysis and Other Mathematics, 1 (2006), pp. 37–46.

[11] J.-L. Thiffeault, *Braids of entangled particle trajectories*, Chaos, 20 (2010).