Computer-Based Visualizations and Manipulations of Matching Paths

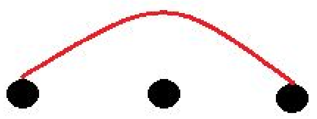
Valerie Zhang

Abstract

Given n points in the 2-D plane, a matching path is a path that starts at one of these n points and ends at a different one without going through any of the other n - 2 points. Matching paths, as well as an important operation called the Hurwitz move, come up naturally in the study of complex algebraic varieties. At the heart of the Hurwitz move is the twist operation, which "twists" one matching path along another to produce a new (third) matching path. Performing the twist operation by hand, however, is not only tedious but also prone to errors and unnecessary complications. Therefore, using computer-based methods to represent matching paths and perform the twist operation makes sense. In this project, which was coded in Java, computer-based methods are developed to perform the twist operation efficiently and accurately, providing a framework for visualizing and manipulating matching paths with computers. The computer program performs fast computations and represents matching paths as simply as possible in a simple visual interface. This program could be utilized when solving open problems in symplectic geometry: potential applications include characterizing the overtwistedness of contact manifolds, as well as better understanding braid group actions.

1 Introduction

Given n points in the 2-D plane, a matching path is a path that starts at one of these n points and ends at a different one without going through any of the other n - 2 points. Matching paths are considered up to deformation (see Section 2.1), allowing them to be represented as discrete data[5]. For instance, the two seemingly different matching paths in Figure 1(a) and Figure 1(b) actually represent the same matching path.


(a) Path

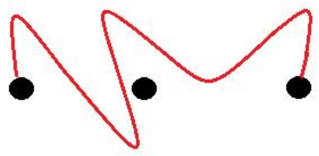

(b) Another Diagram of the Same Path

Figure 1: Matching Paths Up to Deformation

Solution sets of polynomial equations in complex variables are important objects to study [1]-[7]. These solution sets have natural symplectic manifold structures[3], and there has been interest in studying these properties[1],[8].

One strategy to understand a symplectic manifold is to encode it by some combinatorial data. A diagram of an ordered collection of matching paths determines a symplectic manifold [4],[7]. It is crucial to note that some diagrams correspond to the same manifold. The Hurwitz move (see page 4 of [4] for a definition), which acts on a collection of matching paths[4], is a way to generate such diagrams.

Given a solution set of a polynomial equation, there is an algorithmic method to generate a diagram to represent the solution set as a symplectic manifold[8], [9]. For instance, the solution set to the equation $(xy^2 - 1)x - z^2 = 0$, where x, y, and z are complex variables, can be expressed as the diagram shown in Figure 2[8], [9] . This diagram, however, may be unnecessarily complex, and the Hurwitz moves can potentially be utilized in order to generate a simpler diagram[8],[9].

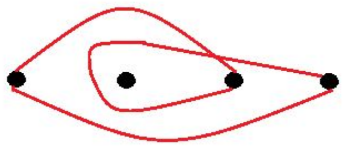


Figure 2: Diagram Representing Solution Set

Not only are twist operations — and consequently Hurwitz moves — hard to perform by hand (see Section 3.6 for a possible output), but the process is also prone to errors. Additionally, performing the operation by hand may yield unnecessarily complex diagrams. A computer-based method of performing the twist operation, the core procedure of the Hurwitz move, could be useful because it not only represents and manipulates matching paths, but also generates the simplest possible diagrams through "canceling out" unnecessary twists. Thus, the program aims to maximize convenience for the user.

The program prompts the user for the input, two diagrams of matching paths, outputting a diagram of the new matching path resulting from the twist operation. The program considers deformations of matching paths (see Section 2.1), translating the geometric Hurwitz move problem into an algebraic problem.

Using two different kinds of discrete data, the computer breaks the twist operation into a finite number of simpler twists. The computer analyzes how these simpler twists affect one type of the discrete data by referring to a repertoire of formulas. Building the repertoire, where the cancellations occur, was the hardest part of the project because the tedious casework required accuracy and effort. After all of these simpler twists are performed, the computer obtains the data of the matching path resulting from the twist operation and produces a diagram of the result.

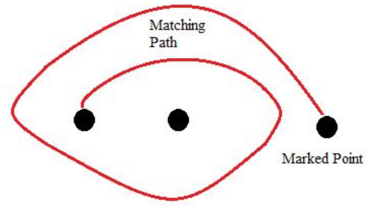
Section 2 provides a technical background for the paper, reviewing basic definitions (section 2.1), operations (section 2.2), and representations (section 2.3) of matching paths. Section 3 describes the implementation steps of the computer-based method and work through an example. The paper concludes by providing potential applications and suggesting further directions for the work.

2 Background And Motivation

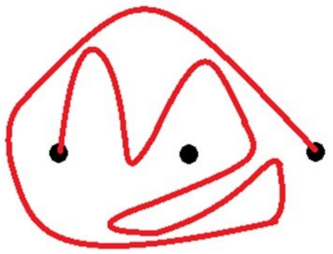
In this section, basic terms, such as "marked point", are defined[6]. Examples will then be provided to explain the concepts, and the importance of visualizing and performing the twist through the computer will also be highlighted.

2.1 Marked Points, Matching Paths, and Isotopic Paths

Take n points in the $2D$ plane. These points are called "marked points" and are denoted by fully filled dots. For instance, Figure 3(a) below has 3 marked points. A matching path is a path that starts at one of these points and ends at another point without going through any of the other remaining $n - 2$ points. Figure 3(a) shows a matching path which starts at the leftmost point and ends at the rightmost point, crossing over neither itself nor the middle marked point. For convenience, matching paths will be simply referred to as "paths."



(a) Three Marked Points and One Matching Path



(b) Path Resulting from the Twist

Figure 3: Isotopic Path

Two paths are isotopic if one path can be deformed into the other without crossing over a marked point. Figure 3(b) shows a path isotopic to the one in Figure 3(a) because either path can be deformed into the other without crossing a marked point. This paper will consider only diagrams — an ordered set of paths through a number of marked points — with co-linear marked points, as any diagram can be deformed into an isotopic diagram with only co-linear marked points.

2.2 Twist Operations

The heart of the Hurwitz move is the twist operation[5],[6], which acts on a path α, with respect to another path β, to produce a new path, γ. The path γ, which is created by the twist operation, differs from α only within a specified region of interest. The region of interest is a small neighborhood around β. The neighborhood can be made as small as possible, as the exact neighborhood chosen does not matter because paths are considered up to deformation.

Figure 4 shows an example of the twist, which deforms the original path (Path 1 in blue) with respect to another path (Path 2 in black), producing a new path (Path 3 in red). The new Path 3 (red) differs from the original Path 1 (blue) only within the largest circle, which denotes the region of interest defined by Path 2 (black). This region is filled with a series of radially expanding concentric circles centered at the middle of the two black marked points of Path 2 (black). In other words, the original Path 1 (blue) rotates only within the area enclosed by the largest circle. Heuristically, the new Path 3 (red) is created by the intersections of rotating Path 1 (blue) with these radially expanding concentric circles. Note that the neighborhood is not always a disk: it is only a region around the path one is twisting with respect to. This neighborhood, however, can be deformed into a circular neighborhood.

Within the green region enclosed by the circle containing the two black marked points, the angle of rotation increases from 0 degrees to 180 degrees. This region will be referred to as the region of increasing degree (green). Within the yellow region that covers the rest of the region of interest, the region of decreasing degree (yellow), the angle of rotation decreases from 180 degrees

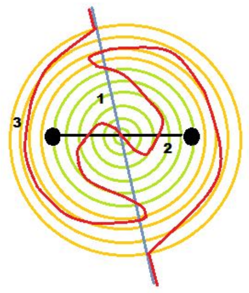


Figure 4: Example of Twist

to 0 degrees. The vertex of the angle of rotation is at the midpoint of Path 2 (black), and measured, counterclockwise as positive, from the original position of Path 1 (blue) before twisting.

The detailed steps are described as follows:

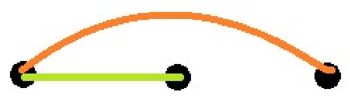
1. Creating the region of interest (green and yellow): Start by considering the circular region around the path (Path 2 in black) one is twisting with respect to. Draw a number of concentric circles centered at the middle of the Path 2 (black), ensuring that the two marked points are on the same circle. Note that the region of interest is determined by a single path (Path 2 in black), and that it extends beyond the region of increasing degree (green), which consists of all circles with diameters no more than the length of Path 2 (black).

2. Rotating within the Region of Increasing Degree (green): Start from the innermost circle. Slowly rotate the path to be twisted (blue Path 1) counterclockwise by increasing angles along these circles. At the center, rotate by 0 degrees. Continuously increase the angle of rotation until reaching the circle containing the marked points, where the rotation angle is 180 degrees.

3. Rotating within the Region of Decreasing Degree (yellow): In this region, denoted by the yellow circles, the rotation angle decreases from 180 degrees to 0 degrees. Starting from the circle containing the two marked points, rotate by 180 degrees. Continuously decrease the angle of rotation until reaching the outermost yellow circle, where the rotation angle is 0 degrees.

Note that because only the portion within the circular region of Path 1 (blue), not the whole of Path 1 (blue), is rotated with respect to Path 2 (black), everything outside the region of interest remains unchanged.

It is possible to twist a straight path with respect to a non-straight path. Figure 5 shows an example where the straight, green path in Figure 5(a) is twisted with respect to the non-straight path (orange) to produce the red path in Figure 5(b).


(a) Original Path

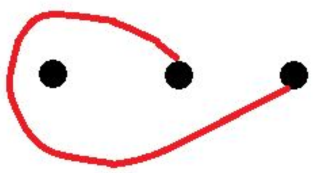

(b) Path Resulting from the Twist

Figure 5: Twist with respect to Non-Straight Path

Additionally, the twists described in this section can be "canceled out" or "untwisted". The inverse of a twist is realized when a path is rotated in the clockwise direction rather than the counterclockwise direction, or vice versa. The inverse twist "cancels out" the twist, as shown in Figures 6 and 8. The untwisted path in Figure 8 is isotopic to the original path in Figure 6, and Figure 7 shows an intermediary step in the deforming of one path into the other.

If the non-straight path (Path 1 in red) in Figure 6 is first twisted with respect to the straight line (Path 2 in green) and the inverse operation of the twist with respect to Path 2 (green) is then performed, Path 3 (red) in Figure 8 will be obtained. Path 3 looks more complicated from Path 1 in Figure 6, even though they are isotopic. This undesirable issue of over-complication occurs when performing the twist geometrically, but the computer-based approach developed resolves this issue by returning the simplest diagram of the resulting path.

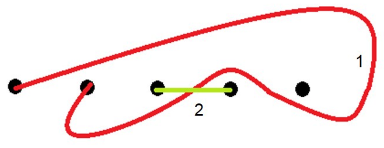


Figure 6: Original Path

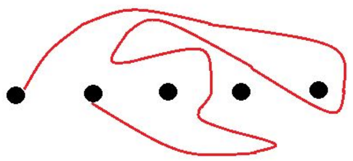


Figure 7: Deformed Original Path

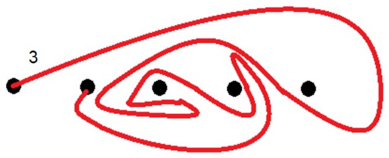


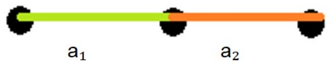
Figure 8: Untwisted Path

2.3 Representations of Paths

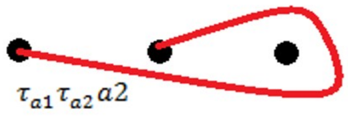
Three different ways to represent paths are used: the twist word data, the single slit data, and the double slit data. These three types of path representation will enable the computer-based twist operations.

2.3.1 Twist Word Data Representation

One way to represent paths is by expressing them in terms of twists performed with respect to the straight line paths, or the "basic paths," connecting adjacent marked points. For instance, there are two basic paths in Figure 9(a), a_1 and a_2.



(a) Original Path



(b) Untwisted Path

Figure 9: Classifying Twist with Twist Word Data

The twists performed with respect to a basic path are known as the "basic twists." A "twist word" of a path is notated as follows:

- Twist: τ
- Inverse Twist: τ^{-1}
- Twist path b with respect to path a: $\tau_a b$

By convention, the direction of a twist is counterclockwise. Thus, the direction of an inverse twist is clockwise. Any path can be obtained from a single basic path by performing multiple twists with respect to basic paths. The path shown in Figure 9(b) can be represented by a twist word of $\tau_{a_2}\tau_{a_2}a_1$. Read from right to left, the twist word indicates that the path shown in Figure 9(b) is obtained by twisting a_1 with respect to a_2, and twisting the result once more with respect to a_2. Note that when finding the twist word data of a path, it suffices to "untwist" the path into a single basic path, as the twist word of the original path will simply be the inverse of the word obtained by "untwisting."

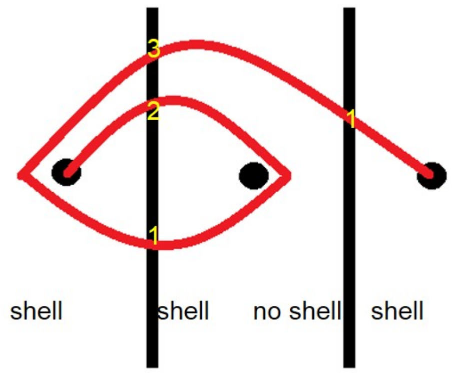
Given two matching paths, α and β, as well as the twist word data for α, it is possible to express $\tau_\alpha\beta$ in terms of basic twists performed with respect to β. The lemma below details the formula for the resulting twist and will be the basis for finding the twist word in the computer. In practice, the lemma can be applied iteratively, until β eventually becomes a basic path.

Lemma. *Given two matching paths, α and β, $\tau_\alpha\beta$ can be expressed in terms of β and the twist word for α. Suppose $\alpha = \tau_\gamma\omega$ and γ is a series of basic twists applied to ω to obtain α. Then $\tau_\alpha\beta = \tau_\gamma^{-1}\tau_\omega\tau_\gamma\beta$.*

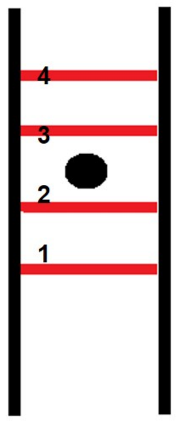
The two paths in Figure 5(a) can provide an example of how the lemma is used. Suppose the two basic paths in Figure 5(a) are b_1 and b_2. Then, the twist word for the green path is simply b_1, and the twist word for the orange path is $\tau_{b_2}^{-1}b_1$. Applying the formula, the path resulting from the twist will be $\tau_{b_2}\tau_{b_1}\tau_{b_2}^{-1}b_1$, which is the path displayed in Figure 5(b). Thus, the lemma holds true in this case.

2.3.2 Single Slit Data Representation

Another way to represent paths is the single slit data representation [5]. Black vertical lines are drawn between every two adjacent marked points, as demonstrated in Figure 10(a). The path is then deformed until it intersects each vertical line a minimal number of times. The region between two adjacent vertical lines is called a slit. The two outermost regions that extend infinitely are also treated as slits.



(a) With Shell



(b) No Shell

Figure 10: Single Slit Data

The intersections of the red path with each black vertical line is numbered bottom up, as seen in Figure 10(a). The number of intersections in Figure 10(a) are 3 and 1, from left to right, respectively.

A shell refers to the part of a path that starts at a given vertical line and returns to that same vertical line before crossing any other vertical line. A slit with an end point is also considered to have a shell. One can number the intersection of the path with each vertical line and return the indices for the largest shell, or the outermost shell, of a slit. One can also determine whether there is a shell on each side of the vertical lines. Figure 10(a) shows an example of a path with shells, intersections numbered. Figure 10(b) shows a single slit with no shell. In such slits without shells, it suffices to give the index of the first line that goes below the marked point. This is called the "first down" of a slit, and only slits that do not extend definitely can possibly have a first down. In Figure 10(b), the second intersection is the uppermost line to go beneath the marked point, so the first down value is 2. If a slit has no shells and the lines only go above the marked point, then the first down value of the slit is -1.

Single slit data determines unique paths. In detail, the components of single slit data are:

- Number of intersections with vertical lines between slits
- Existence of shells, also known as shell values. Ones and zeros to used to signify if there exists a shell or not, respectively. For instance, the shell values for Figure 10(a) are $< 1, 1, 0, 1 >$.
- Indices of largest shell, referred to as largest shell values. For Figure 10(a), the largest shell values are $[1, 3]$, $[1, 2]$, null, and $[1, 1]$, where null refers to there not being a shell to the left of the second vertical line. Additionally, because endpoints are considered to be shells, the largest shell of the rightmost slit is $[1, 1]$.
- Down data. For Figure 10(a), the down data would be [null], because the leftmost and rightmost slits are not considered, and the middle slit has shells. For each slit with shells, the down data value is null.

2.3.3 Double Slit Data Representation

Another way to represent paths is through the double slit data representation, which looks at the region of the path around two adjacent slits. Double slit data representation can be thought of as an intermediary between twist word data representation and single slit data representation. In this paper, this representation is used to see how the single slit data of a path changes when it is twisted with respect to a basic path. A double slit region of any arbitrary, non-basic path can be expressed in terms of the five types displayed in Figure 11 (courtesy of [5], Figure 15). Type *IV* is the special case that occurs when the path is a basic path.

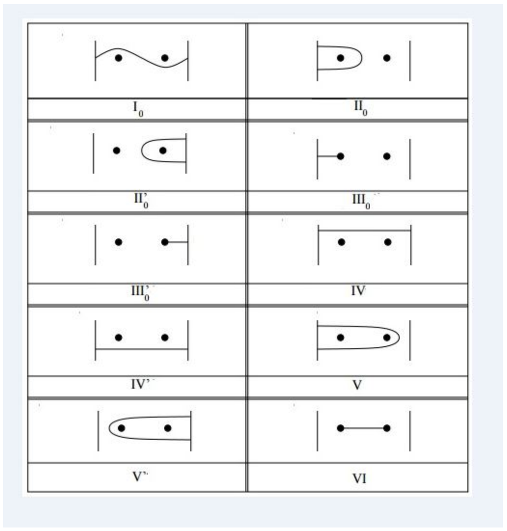


Figure 11: Basic Types of Double Slit Data

The segments of type I_0, II_0, II'_0, III_0 and III'_0 in Figure 11 can be repeatedly twisted in the clockwise or counterclockwise direction with respect to the corresponding basic path to obtain more complicated subsections. Twisting in the counterclockwise direction will increase the subscript of the type by one, while twisting in the clockwise direction (an inverse twist) will decrease the subscript by one. For instance, type I_3, which is obtained by twisting I_0 counterclockwise with the basic path three times, is shown in Figure 12 (courtesy of [5], Figure 16).

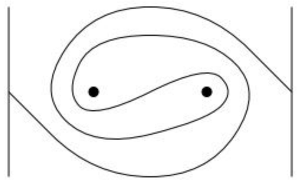


Figure 12: Type I_3 Generated by Twisting I_0 Counterclockwise Three Times

Now, one can express each double slit region in terms of the subsections in the region. For instance, the double slit region in Figure 13 is composed of one II_1 segment, one I_0 segment, and two IV' segments. Note that the properties of isotopy also apply to these subsections of the path, deforming the double slit region without crossing over marked points does not change its representation. Additionally, since the twist operation is expressed as only a change in the subscript of the double slit data, the double twist data representation is a convenient method to describe the twists.

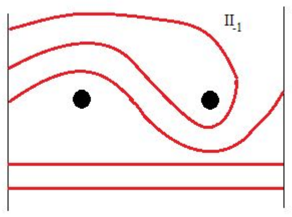


Figure 13: Example of Double Slit Region

3 Computer-Based Path Manipulation

The main goal of this computer-based methodology is to perform the twist. Given two matching paths, the Java program twists one path with respect to the other and returns the result in the simplest possible form. To do so, one has to first input paths to the computer. Then, the computer finds the single slit data representation for the given path and is able to convert the single slit data to twist word data. Using the twist word data and the repertoire of formulas describing how basic twists affect single slit data, the computer performs the twist and calculates the single slit data of the resulting path. Finally, the computer produces a diagram of the resulting path. Figure 14 shows of flowchart of the process for twisting a path X with respect to a path Y. This section will explain each task in detail by working through an example: starting from two matching paths in Figure 16 as input, the program produces the simplest picture of the path created from the twist.

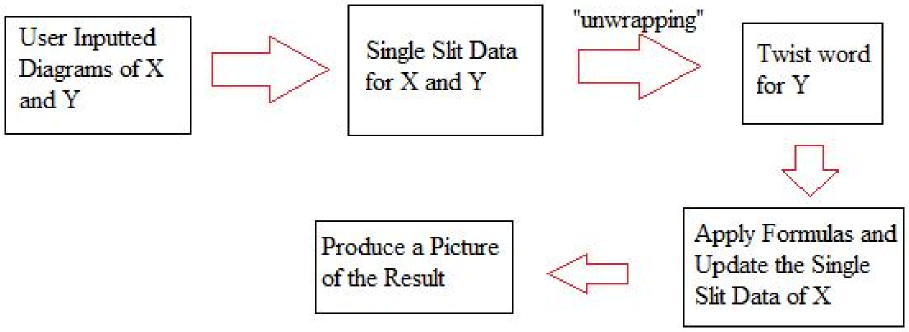

User Inputted Diagrams of X and Y
Single Slit Data for X and Y
"unwrapping"
Twist word for Y
Apply Formulas and Update the Single Slit Data of X
Produce a Picture of the Result

Figure 14: Flowchart of Computer-Based Process

3.1 Inputting Paths to the Computer

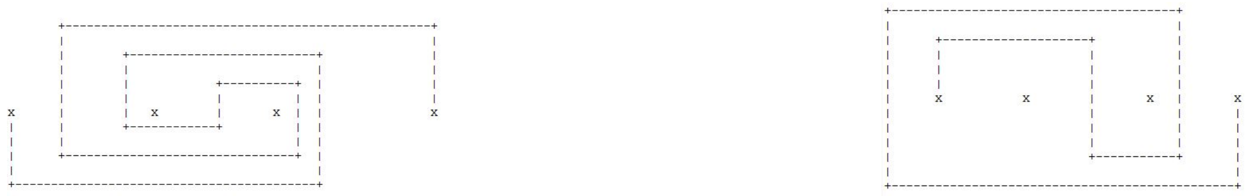
The first step is inputting the path to the computer. The path is represented with “x”, “+”, “-”, and “|” characters, as seen in Figure 15. The “x” characters are marked points, the “+” characters refer to changes in direction, the “-” characters indicate parts of the path that are largely horizontal, and “|” characters signify parts of the path that are largely vertical. Note that the plus signs indicate the existences of shells, as the path changes directions only at shells. In Figure 15, the path initially goes vertically, and the plus sign specifies the change to the horizontal direction, indicating the existence of a shell in the leftmost slit.

```
+---------------------------------+
|                                 |
|                                 |
x          x              x       |
           |                      |
           |                      |
           +----------------------+
```

Figure 15: Path Representation in Computer

This section will work through the program’s twisting of the path in Figure 16(b) with respect to the path in Figure 16(a). For convenience, the path in Figure 16(a) will be referred to as “Path A,” and the path in Figure 16(b) will be referred to as “Path B.”

With this representation, it is easy to see where the marked points are, and the rest of the elements of the single slit data can also be found somewhat easily.



(a) Path A (b) Path B

Figure 16: Path Examples

3.2 Finding Single Slit Data From Path Diagrams

The next step is returning the single slit data for a path. This subsection will explain the process in detail, but the reader should feel free to first skip to the computer-generated results in Figure 17 before reading the rest of the subsection.

The program prompts the user for a text file representation of the path as input (see Section 3.1), converting the characters in the file (including spaces) into a matrix. From there, it calculates the single slit data.

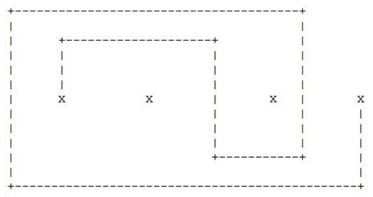
To find the number of intersections, the program first divides the matrix into slits by storing all locations of “x”s, which signify marked points. Next, the program looks at two columns between two adjacent marked points. For both columns, the program looks for the number of “-” characters, and the number of intersections is the minimum of those two values.

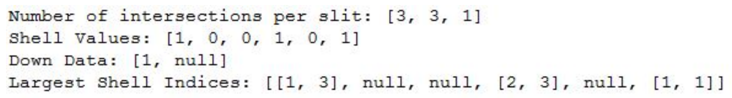
To find the shell values, the program uses the fact that “+” characters signify the existence of shells, as mentioned before. To see if there is a shell to the left, the program starts at a slit’s rightmost edge and searches leftwards for the “+” character, until the next marked point is reached. The program returns a 1 if there is a “+” character, and 0 if not. Similarly, the program starts at a slit’s leftmost edge and searches rightwards for the “+” character to see if there is a shell to the right.

The program only finds the largest shell values for regions that correspond to a “1” in the shell value data. For regions corresponding to a “0”, the program returns “null” as the largest shell. To find the largest shell index to the left, the program searches each row of the slit for “+-”, which signifies the edge of a shell. The program stores the row indices of all rows containing “+-” in an ArrayList. The largest shell value is found by returning the positions of the first and last entries in the ArrayList. To find the largest shell values on the right, the program searches each row of the slit for “-+”. The largest shell values are then found similarly.

To find the down data, the program examines the columns containing marked points that are in slits with no shells. In these columns, the program searches for “-” characters below the marked point, signifying lines below the marked point. If one or more “-” characters have been found, the program will return the number of “-” characters found. If not, the program searches for “-” above the marked point, returning -1 when “-” has been found and null if otherwise.

Figure 17 shows single slit data result for Path B, shown again in Figure 17(a). Figure 17(b) shows the single slit data calculated by the program.


(a) Path B



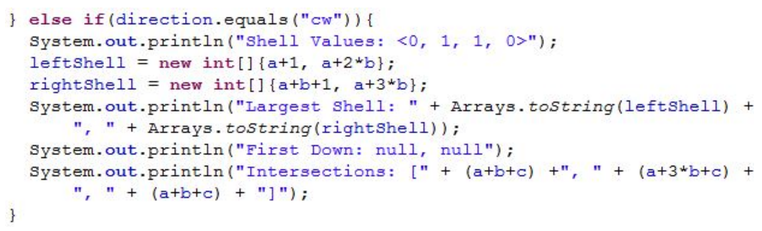
```
Number of intersections per slit: [3, 3, 1]
Shell Values: [1, 0, 0, 1, 0, 1]
Down Data: [1, null]
Largest Shell Indices: [[1, 3], null, null, [2, 3], null, [1, 1]]
```

(b) Single Slit Data for Path B

Figure 17: Single Slit Data Results

3.3 Evaluating How Basic Twists Affect Single Slit Data

Rather than performing the twist geometrically, another method is used to determine how the single slit data changes each time the given path is twisted with respect to a specified basic path. Because performing a basic twist only affects the single slit data of the two adjacent slits containing the basic path, it suffices to look only at those two slits, which will be called the double slit region. The program is coded such that it splits the double slit regions into cases based on its shell values and its double slit data. From there, it finds and applies the corresponding formulas in the repertoire, returning the new single slit data of the double slit region after the twist. This repertoire contains formulas for how every possible double slit region changes based on whether a twist or an inverse twist is performed, enabling the performance of the twist operation. Figure 18 shows part of the code for the formula.

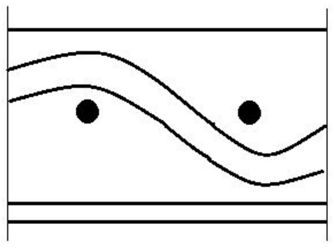


```
} else if(direction.equals("cw")){
  System.out.println("Shell Values: <0, 1, 1, 0>");
  leftShell = new int[]{a+1, a+2*b};
  rightShell = new int[]{a+b+1, a+3*b};
  System.out.println("Largest Shell: " + Arrays.toString(leftShell) +
      ", " + Arrays.toString(rightShell));
  System.out.println("First Down: null, null");
  System.out.println("Intersections: [" + (a+b+c) +", " + (a+3*b+c) +
      ", " + (a+b+c) + "]");
}
```

Figure 18: Partial Code for Formula Applied to Path A

For example, consider the two cases that occur when the double slit region has shell values of 0,0,0,0: the region contains either I_0 segments or I_1 segments but not both. Figure 19(a) and Figure 19(b) show two examples of the two different cases.

The code starts from the single slit data of the double slit region and computes double slit data of the corresponding region. From here, the program determines the updated single slit data based


(a) 0000 slit containing I_0

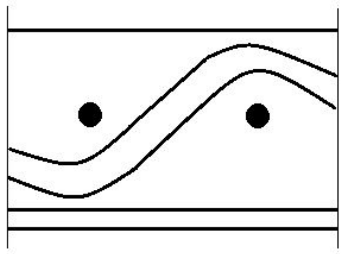
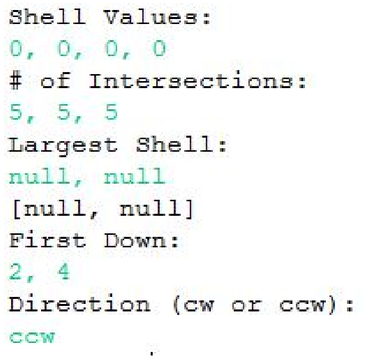

(b) 0000 slit containing I_1

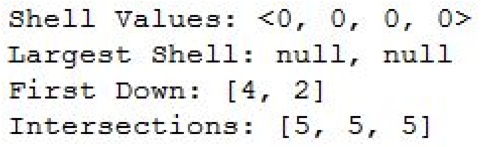
Figure 19: Double Slit Regions

on the number of IV, IV', I_0, and I_1 segments in the double slit region. For instance, Figure 19(a) was twisted counterclockwise with respect to the corresponding basic path. Figure 20(a) shows the user-generated input of the single slit data of Figure 19(a). The black directions are prompts from the computer, and the green font is information entered by the user. Figure 19(b) is the diagram resulting after the twist, and Figure 20(b) shows the corresponding single slit data calculated by the computer. The reader can confirm that the diagrams Figure 20(b) matches up with Figure 19(b).



```
Shell Values:
0, 0, 0, 0
# of Intersections:
5, 5, 5
Largest Shell:
null, null
[null, null]
First Down:
2, 4
Direction (cw or ccw):
ccw
```

(a) Single Slit Data as Input



```
Shell Values: <0, 0, 0, 0>
Largest Shell: null, null
First Down: [4, 2]
Intersections: [5, 5, 5]
```

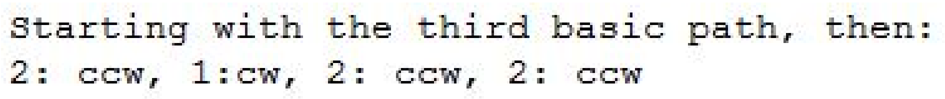
(b) Resulting Single Slit Data After Twist

Figure 20: Source Code for Single Slit Data

3.4 Obtaining the Twist Word Data for a Path

In this step, the program determines the twist word for the given path by incrementally "untwisting" the path. The program examines each double slit region one at a time, starting from the leftmost double slit region. If twisting with respect to the basic path clockwise or counterclockwise will simplify the double slit region, the program performs the twist and updates the single slit data, returning to the leftmost region again to repeat the process. The new single slit data is calculated by using the methods described in Section 3.3. If twisting in either direction will not simplify the region, the program moves to the next double slit region.

If the program examines every double slit region of the path without performing any twists, then path has been fully untwisted, and the twist word will be returned, in the form of the basic paths and the directions twisted with respect to. For instance, the twist word data representation of Path A, shown in Figure 16(a), is represented in Figure 21. The directions printed by the program specify that the twist word of Path A is $\tau_{a_2}\tau_{a_2}\tau_{a_1}^{-1}\tau_{a_2}a_3$, where a_1, a_2, and a_3 are the basic paths.

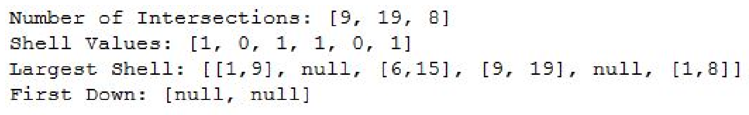


```
Starting with the third basic path, then:
2: ccw, 1:cw, 2: ccw, 2: ccw
```

Figure 21: Twist Word Data for Path A

3.5 Calculating Single Slit Data for the Resulting Path

Now that the twist word data of the path has been determined, one can perform the twist operation. The program utilizes the formula in Lemma in conjunction with the repertoire of formulas described in Section 3.3. From here, the single slit data of the resulting path is returned. For instance, by twisting Path B with respect to Path A, one can obtain the single slit data of the path shown in Figure 22.



```
Number of Intersections: [9, 19, 8]
Shell Values: [1, 0, 1, 1, 0, 1]
Largest Shell: [[1,9], null, [6,15], [9, 19], null, [1,8]]
First Down: [null, null]
```

Figure 22: Single Slit Data of Resulting Path

3.6 Generating a Diagram of the Resulting Path

Given the single slit data of the path resulting from the twist operation, the program creates a diagram of it using vertical and horizontal lines. The program draws the path one slit at a time. To do so, the program first determines the left and rightmost bounds for each slit. Slits without shells have a width of 80 pixels. The width for slits with shells is determined by the expression $30 + 30s$, where s is the number of shells in the slit, not including the endpoints.

Next, slits are drawn one by one. For slits without shells, the program finds the number of intersections and draws that number of horizontal lines, spacing them evenly. In slits with shells, the program first draws all the horizontal lines above the shell. Next, the shells are drawn, the outermost shells first and the innermost shells last. Finally, all horizontal lines below the shell are drawn. Figure 23 provides the diagram of the path resulting from the twist operation. The red squares denote marked points.

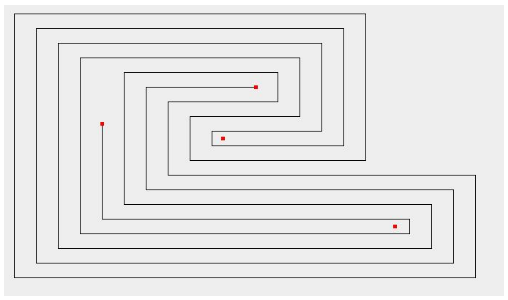


Figure 23: Diagram of Resulting Path

4 Conclusion and Future Work

This paper developed a computer-based methodology to visualize and manipulate matching paths, enabling the performance of the twist operation both efficiently and accurately. It relied on a repertoire of formulas that determined how applying basic twists affected the single slit data of a path. The computer-based approach is a promising way to perform the Hurwitz move and return the simplest possible diagram of the resulting path, thus eliminating unnecessary complexities encountered in solving complex equations.

The program's visual representation of paths can be improved. The current input method is limited to a vertical and horizontal line representation to capture the basic essence of the paths. Future work can be done to make the input method less restrictive, as well as modifying the diagrams produced by the computer such that the marked points are co-linear. This modification is not difficult: it requires only translating parts of the diagram by a specified amount.

This computer-based methodology provides a framework for representing paths and performing operations, making possible fast computations through a simple graphical interface. It sheds light on the usage of computerized methods in symplectic geometry. Potential applications include characterizing the overtwistedness of contact manifolds[2], as well as better characterizing braid group actions[5].

5 Acknowledgements

I would like to extend my greatest appreciation to my mentor, Umut Varolgunes, for introducing me to the fascinating world of Hurwitz moves and matching paths, as well as all the insightful ideas to guide me along the way. I would also like to thank the MIT PRIMES program for facilitating this research and providing the venue to share research findings. Furthermore, I would like to thank Dr. Jadrian Miles for coding tips and time. I would also like to extend my thanks to Dr. Tanya Khovanova for guiding many PRIMES projects. Finally, I would like to thank my parents for all the encouragement and support.

References

[1] V.I. Arnold. *Some Remarks on Symplectic Monodromy of Milnor Fibrations*, volume 133 of *Progress in Mathematics*. Birkhäuser Basel, 1995.

[2] R. Casals, E. Murphy, and F. Presas. *Geometric Criteria for Overtwistedness*. 2015. Submitted for Publication.

[3] Y. Eliashberg and M.Gromov. Convex Symplectic Manifolds. *Proceedings of Symposia in Pure Mathematics*, 52(2), 1991.

[4] E. Giroux and J. Pardon. *Existence of Lefschetz Fibrations on Stein and Weinstein Domains*. 2016. Geometry Topology (Accepted for Publication).

[5] M. Khovanov and P. Seidel. Quivers, Floer Cohomology, and Braid Group Actions. *Journals of the American Mathematical Society*, 15:203–271, 2002.

[6] M. Reid. *Undergraduate Algebraic Geometry*. Cambridge University Press, 1989.

[7] P. Seidel. A Biased View of Symplectic Cohomology. *Current Developments in Mathematics*, pages 211–253, 2008.

[8] P. Seidel. Fukaya Categories and Picard Lefschetz Theory. *Bulletin of the American Mathematical Society*, 47(4):735–742, 2010.

[9] P. Seidel. Picard–Lefschetz Theory and Dilating C*-actions. *Journal of Topology*, 8(4):1167–1201, 2015.