

Katsumi Cook & Grace McClurg: Determining the Braid Index of Alternating Links

1. How to run code

We use **Mathematica Version 11.3**.

Execute **Evaluate Initialization Cells**.

Execute the function **homeScreen[]** at the beginning of the file. This will open a window with the start menu of the user interface.

Click the “Calculator” button to be taken to a page which allows you to choose between four general types of links to input.

After entering each link, close the window and re-evaluate **homeScreen[]** to enter a new link.

Test cases to demo (additional information about output, which was omitted in the hard copy, has been included below in red):

Unspecified link demo (continued fraction sequences): From the “Calculator” page, go to the “Unspecified Link” page. Choose “View all possible orientations,” then “Fractional or Continued Fraction.” Input the link $\{\{2,2,1\},\{2,2\},\{3\},2\}$. The program will output a list of all possible four-directions sequences for this link. Copy the first list (which will be $\{\{L,R,R,L\},\{R,L,R,L\},\{R,L,L,R\}\}$), then select “Evaluate the link at one of these directions.” Re-enter the sequence and the four-directions list. The program will output the braid index of the link, 6. Click “Additional Information” to see a representation of the Seifert decomposition of the link, the signed vectors for each of its tangles ($\{\{2,2,1\},\{-2,1,1\},\{3\}\}$), its class (B), and the number of components (2).

Two-bridge link demo (Conway): From the “Calculator” page, go to the “Two-bridge Link” page. Select “Conway Notation” and enter the string “[2 6 5 1 2]”. The program will output the braid index of the link, which is 6. Click “Additional information” to see the possible signed vectors (the only possible signed vector is $\{-2,1,5,6,2\}$) and the number of components (1).

M1 link demo (Fractions): From the “Calculator” page, go to the “M1” link page. Select “Fractional or

continued fraction” and enter the list $\{12/19, 2/3, 2\}$. The program will output the braid index of the M1 link (5). Click “Additional Information” to see a representation of the Seifert decomposition of the link, the signed vectors for each of its tangles $(\{-1, 1, 1, 2, 2\}, \{-1, 1, 1\})$, its class (M1), and the number of components (2).

M2 link demo (Conway): From the “Calculator” page, go to the “M2” link page. Select “Conway notation” and enter the string “[4 1;3 1 1;2]”. The program will output the braid index of the M2 link (5). Click “Additional Information” to see a representation of the Seifert decomposition of the link, the signed vectors for each of its tangles $(\{-1, -3, -1\}, \{-1, -1, -3\}, \{-2\})$, its class (M2), and the number of components (1).

2. Description of completed project

Part 1: Project Description

Our project determines the braid index of Montesinos Links, a class of alternating link. The user provides either a continued fraction (representing a two-bridge link) or a set of continued fractions followed by an integer (representing a Montesinos link with the specified number of lone crossings). In addition to the braid index, the user can access further information that includes the equation used to determine the braid index, a description of the equation, and either details about the link’s Seifert circles or the order of positive and negative crossings in the link.

Part 2: Final Product Description

a. Screen Display

The home screen the user is greeted by appears as it does in Figure A, in which they can choose whether to be directed to the calculator page or the instructions page. If the user was to click the instructions page, they would be redirected to a descriptive screen displayed in Figure B, in which the user could scroll to see more details about knot theory and its relevance in our calculator.

Out[]=

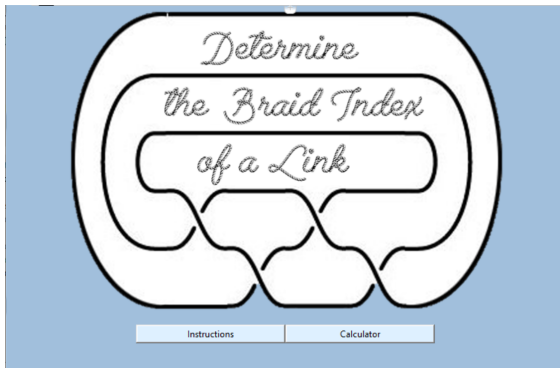


Figure A: Home Screen for User Interface

Out[]=

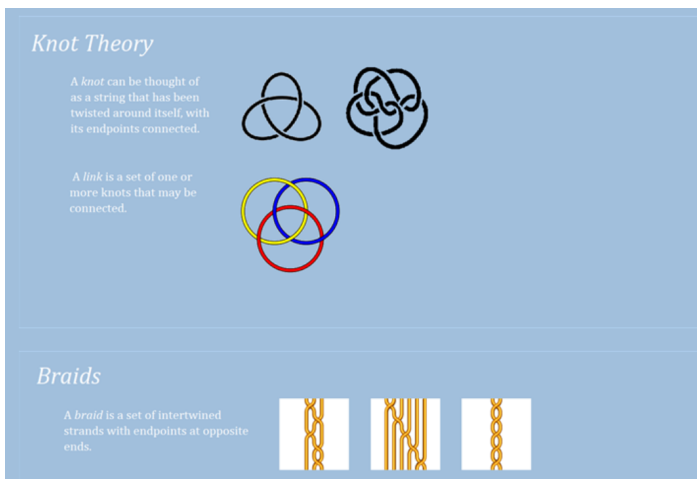


Figure B: Instructions Page

Selecting the calculator button leads the user to a screen in which they can choose whether or not to specify the class of link they are evaluating.

Out[]=

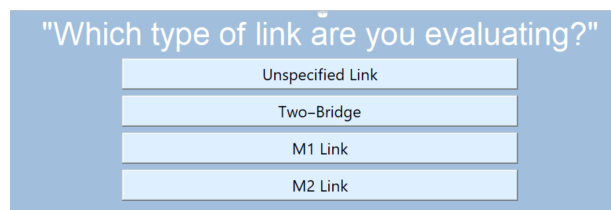


Figure C: This page allows the user to select class of the link they wish to evaluate

If the user was to select an M1 Link, they would be redirected to a page in which they are prompted to choose which notation to specify the desired link in. The notation options include Conway, fractional, and continued fraction form. If the user was to select any of the help buttons, they would be redirected to a page that describes the selected notation in detail.

Out[]=

"Which format is your input in?"

Fractional or Continued Fraction	Help
Conway Notation	Help

Figure D: This page allows the user to specify the notation of their input

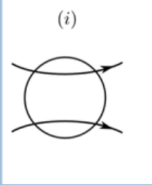
Both of these pages direct the user to calculator pages that appear similar to Figure E, with different labels being present if the user was to indicate the desired notation was Conway. In the first input field, the user can specify the continued fraction or fractional sequence that represents the link. After clicking submit, the braid index appears in the grey rectangle, as it does in Figure E. The calculator interfaces for two-bridge links, M1 links, and M2 links appears in the same structure, with there being only visual differences in regards to labeling.

If the user wanted to learn more about the link they specified in the calculator, they can then click the "Additional Information" page. This page is available on all calculator screens for all classes of links. In the example we illustrated in Figure E, the user would be directed to the page depicted in Figure F. This example page describes the Seifert decomposition of the link, its associated signed vectors, its class, and its number of components. This information is available for a link in any Montesinos class with the exception of two-bridge links. The additional information page associated with two-bridge links can only show its associated signed vectors (there being multiple if it has multiple components) and the number of its components.

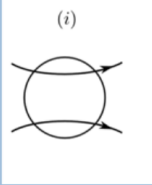
Out[]=

"Seifert Decomposition:"

(i)



(i)



"Signed Vector:"
 $\{-1, 1, 1, 2, 2\}, \{-2, 0, -1\}, 2\}$

"Class:"
"M1"

"Number of Components:"
1

Back to Previous Screen

Figure F: Example of what is displayed when the user requests additional information

Going back to the screen depicted in Figure C, the user could have alternatively choose to not specify the class of the Montesinos link they wished to test. What is distinct about not specifying the link is that a list of the orientations of the four incoming strands at each of the link's tangles is necessary to evalu-

ate its braid index. Because of this, the user is first directed to an options page before entering depicted in Figure G.

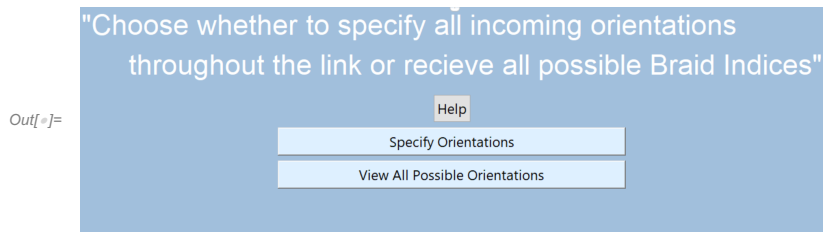


Figure G: Option page for deciding whether to evaluate a link by giving its specified orientations or to first view all possible associated directions with a specified link sequence.

This options page allows the user to decide whether to be directed towards the calculator or to be shown possible sequences of orientations to use in their link evaluation, indicated by the button "View All Possible Orientations". This button redirects the user to a page where they can specify the notation they will represent the link, as shown in Figure D. Once this is specified, the user is directed to a page where all of the possible orientation lists are shown. An example is shown in Figure H, in which there is only one possible direction sequence associated with the specified link. The user can then click the button at the end of the panel to evaluate the specified link with one of the direction sequences by copying and pasting the desired sequence.

In this example, there are two possible directions sequences that can be utilized, and its execution is in conveyed in Figure I.

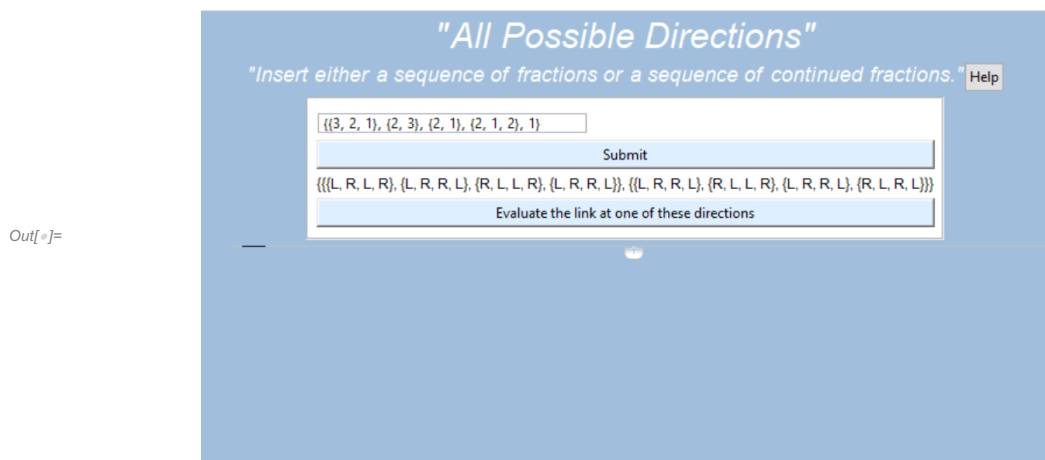


Figure H: An example of all directions associated with this unknown link specified in conway notation

From this point, the user can discover additional information about the link, including its class which they indicated was unknown prior. The additional information page for this example is shown in Figure J

Out[]=

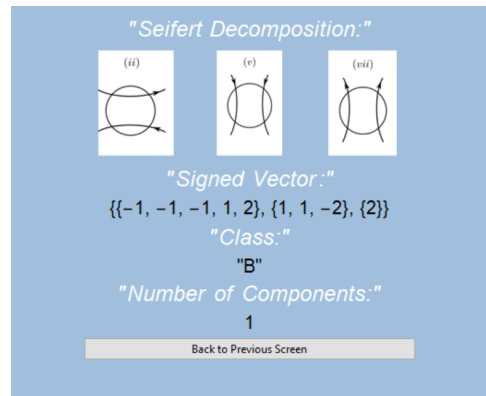


Figure J: This is a figure conveying the additional information associated with a previously unspecified link

Part 2: User Interaction

The user inputs data which represents a Montesinos link. The program outputs the braid index of that link, along with additional information about the link which was used to calculate this number.

To input a two-bridge link, the user enters a continued fraction. The user may enter this in Conway notation (with the stipulation that all numbers in the list must be positive), or in its simplified numerical form (numerator/denominator).

To input a Montesinos link, the user enters a set of continued fractions, followed by the number of lone crossings in the link. The continued fractions may be represented either in Conway notation or in simplified numerical form, as in the two-bridge case.

The program outputs the braid index of the specified link, along with information that was used to calculate the braid index, such as the Seifert parity types of the tangles in the link (with the exception of two-bridge links), the class of the link (with the exception of two-bridge links), the signed vectors associated with the link, the link's class (with the exception of two-bridge links), and the number of the components within the link.

3. Algorithms & Formulas

Increment One: Braid Index of Alternating Two Bridge Links

The objective of our first increment was to create a program to determine the braid index for two-bridge links. We accomplished this by first referencing Dr. Ernst's research paper, "A Diagrammatic Approach for Determining the Braid Index of Alternating Links," which described the equation he and his colleagues determined for calculating the braid index of two-bridge links.

An arbitrary fraction greater than 1 can be “read” according to a set of rules to correspond to a unique two-bridge link. The fraction must be written in continued fraction form, and the integers which are not the 1’s representing reciprocals in the continued fraction are recorded in a list. However, the algorithm for building alternating two-bridge links is not applicable to sequences of even length. In the case that a continued fraction’s sequence is of even length, the last element of the sequence is decremented by 1, and the element “1” is appended to the end of the sequence so that the new sequence has an odd length but represents the same fraction. The function `correctSequence` converts sequences to their appropriate length if they are not already odd.

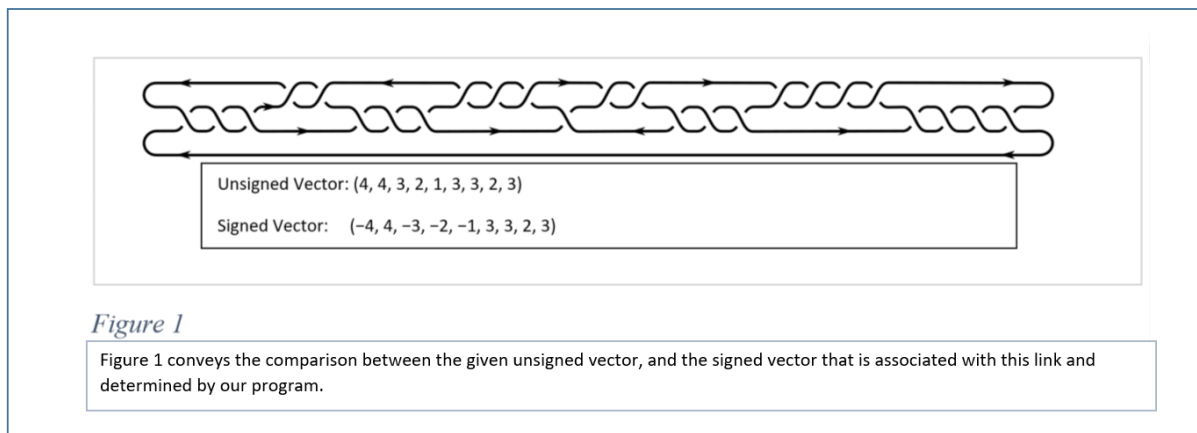
For example, a sequence representing the fraction $\frac{218}{101}$ is {2,6,3,5}, since $2 + \frac{1}{6 + \frac{1}{3 + \frac{1}{5}}}$ is the fraction’s equivalent. However, the example sequence is of even length, meaning its sequence must be corrected. To address this, the fraction is altered to $2 + \frac{1}{6 + \frac{1}{3 + \frac{1}{4+1}}}$, where the last denominator in the fraction, 5, is replaced with the equivalent 4+1. This creates the representative sequence {2,6,3,4,1}, which is of odd length.

Once the sequence representing the two-bridge link is obtained, the link is built by alternating between twisting strands 2 and 3 and strands 1 and 2 of the previous crossing set (see Figure 2) the number of times specified by the current number in the list and making sure that the link is alternating, starting from the right and proceeding left.

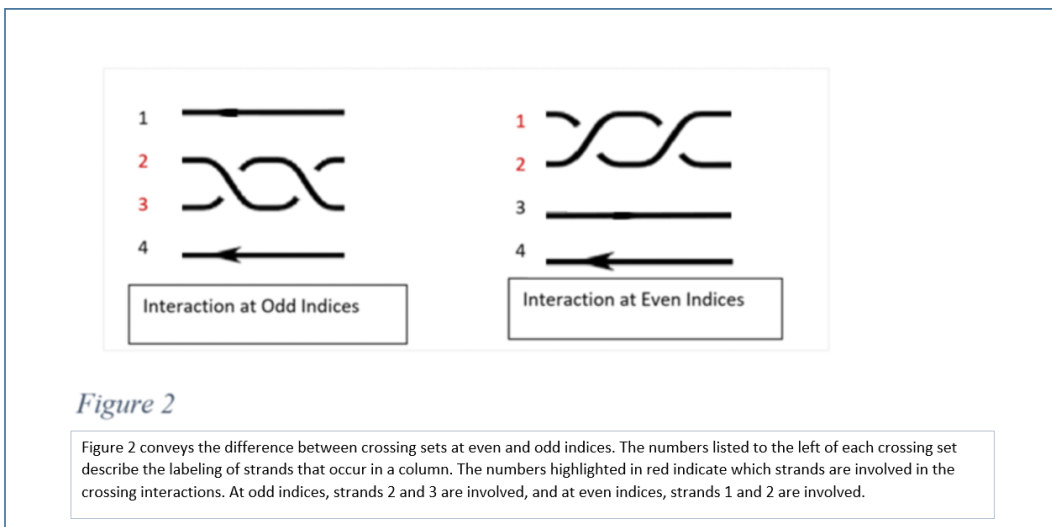
The construction of a two-bridge link can be illustrated with the sequence described earlier, {2,6,3,4,1}. The link is built by performing 2 twists with strands 2 and 3, then using strands 1 and 2 on the left side of the resulting crossing set to build 6 more twists. 3 twists then occur between strands 2 and 3, 4 twists occur between strands 1 and 2, and 1 twist occurs between strands 2 and 3, with each new set of crossings being built to the left of the previous. Strand 3 following the rightmost set of crossings and strand 3 preceding the leftmost set of crossings are connected, and strand 1 following the rightmost set of crossings and strand 1 preceding the leftmost set of crossings are connected to complete the link.

The signed vector for a continued fraction is the sequence of integers described above, with signs appended to each element, corresponding to whether each element represents a set of crossings that is positive or negative. All of the crossings in a single subsection have the same sign; this is conveyed in Figure 1, in which you can clearly see the sign correspondence of each element to the positivity or negativity of each subsection of crossings. When given the signed vector representing a continued fraction, the other functions needed to determine the braid index of a two-bridge link are **sumOfValuesAtIndices** and **braidIndexTwoBridge**. The function `braidIndexTwoBridge` is a general function for the equation for the braid index of a two-bridge link. The function **sumOfValuesAtIndices** is a helper method for this function, which returns the value of one of the two summations in the formula corre-

sponding with the parity of an element's index. However, the given sequence is not signed, and thus the signed vector must be determined by additional functions.

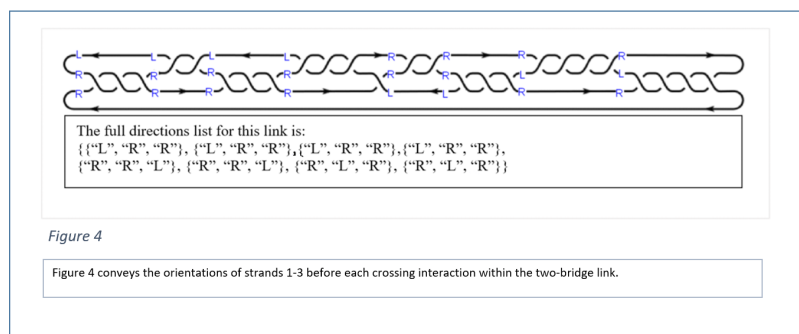
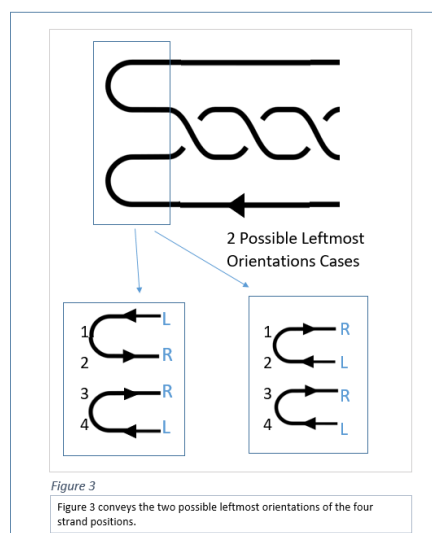


To determine the signs, we must determine the direction of all strands that can potentially be involved in a crossing interaction. The top three strands of the four strands within a column can be involved in a crossing interaction. At even indices, the top strands, 1 and 2, are involved in a crossing interaction. At odd indices, the middle strands, 2 and 3, are involved in a crossing interaction. Figure Two conveys the interactions between strands depending on the index.



All directions in the link depend on the leftmost set of directions, the parity of the index of the current set of crossings, and the parity of the number of crossings in the current set. Our code functions under the convention that strand 4 is oriented right-to-left, represented by direc-

tion “L”, which means that strand 3 is oriented left-to-right (“R”) preceding the leftmost set of crossings. The remaining orientations at the leftmost position to determine are the strands 1 and 2. These strands will either be oriented “RL” or “LR” at the leftmost position because they must have opposite orientations. Note that in Figure 4, strands 1 through 3 are the only strands labeled. This is because our functions are only concerned with the orientations of the top three strands because the orientation of strand 4 will remain constant. Also conveyed in Figure 4 is how we record direction changes after each subsection of crossings. Orientations are recorded before each crossing set.



To determine the orientation of strands, the function **leftmostOrientations** returns the orientations of strands 1, 2, and 3 preceding the leftmost sets of crossings (which is represented by the last integer in the input sequence). The information given by this function is conveyed in Figure 4.

The directions are determined by tracing the strand that begins at position three at the leftmost position around the link and determining its next orientation after each subsection based on the parity of the index currently reached and the parity of the number of crossings in the current set. We will refer to this strand as Origin Strand 3 because it will change positions throughout the two-bridge link. You can see in Figure 5 that Origin Strand 3 occurs at different strand positions throughout the two-bridge link, however, it begins at position three at the leftmost position, before any crossing interactions. The function “**leftmostOrientations**” accomplishes this tracing by utilizing the helper function “**move-**

ntRightLeft", which then is dependent on the helper function "**movement**".

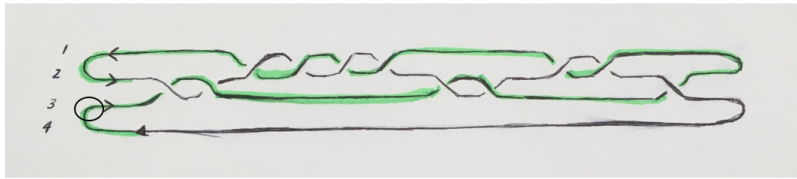


Figure 5

Figure 5 conveys the movement of origin strand 3 throughout the two-bridge link. We trace this movement in our functions to determine the orientations before each crossing interaction.

The function **movementRightLeft** functions by first moving through the link left-to-right, meaning incrementing through the sequence and evaluating the vector values in forwards order, then right-to-left, meaning decrementing through the sequence and evaluating the vector values in backwards order. The function **movement** evaluates the position of Origin Strand 3 depending on the parity of the index reached and the parity of the number of crossings in the current set. Figure 3 conveys the significance of the parity of the index reached as the index determines which strands are involved in the crossing interaction. The parity of the number of crossings determines whether Origin Strand 3 will change positions at all. The 8 different possible position changes for Origin Strand 3 are recorded in Figure Six, in which Origin Strand 3 is highlighted in red.

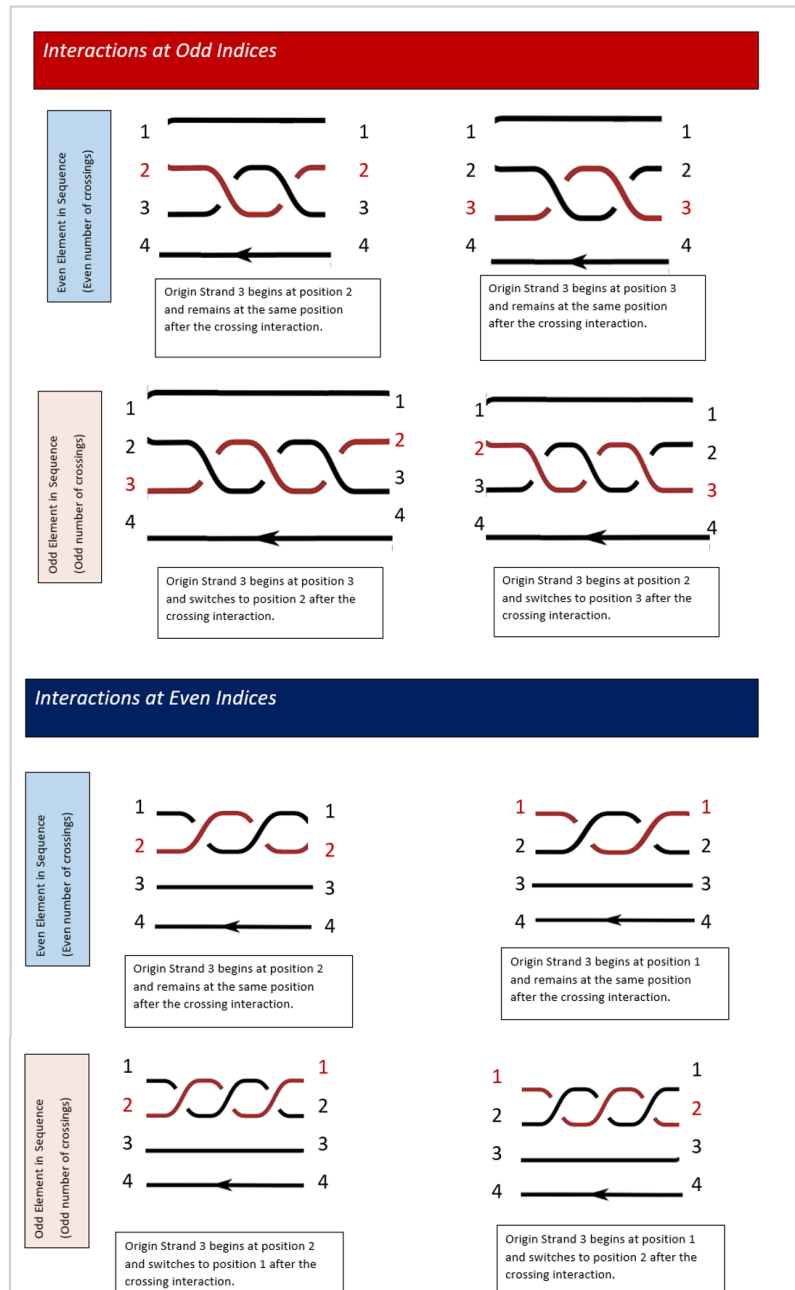
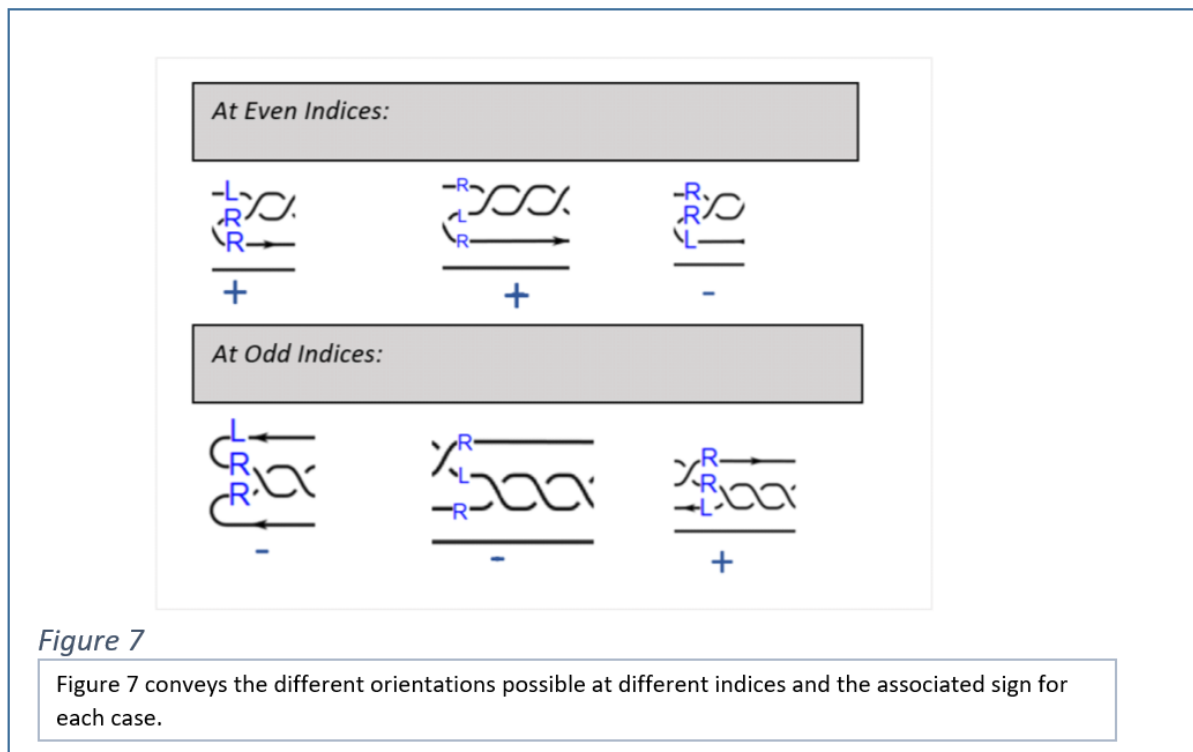


Figure 6

Figure 6 conveys the possible position movements that can occur for Origin Strand 3.

Once the left-most orientation is determined, the **directions** function can determine the orientations of the strands preceding all other sets of crossings in the link, in order from left to right. The **directions** method evaluates the directions of each subsection by determining whether the strands switch places on the basis of the parity of the number of crossings in the last subsection. Figure 7 conveys the possible effects the previous grouping of crossings has on the strand position preceding the current grouping of crossings. The **toSignedVector** method uses this information to determine the signs of each set of crossings, and returns the signed vector form of the input sequence.

In the case of a link, where one of the components may be oriented in two different ways, **toSignedVector** uses the parameter **directionsChoice** to specify the choice of left-most orientation, “LRR” or “RLR”. The six different sign cases the function **toSignedVector** evaluates are illustrated in Figure Seven.



The function **braidIndexTwoBridge** returns the braid index for a two-bridge link using the signed vector which is determined by the functions described above.

Increments Two and Three: Braid Index of Alternating Montesinos Links

Background

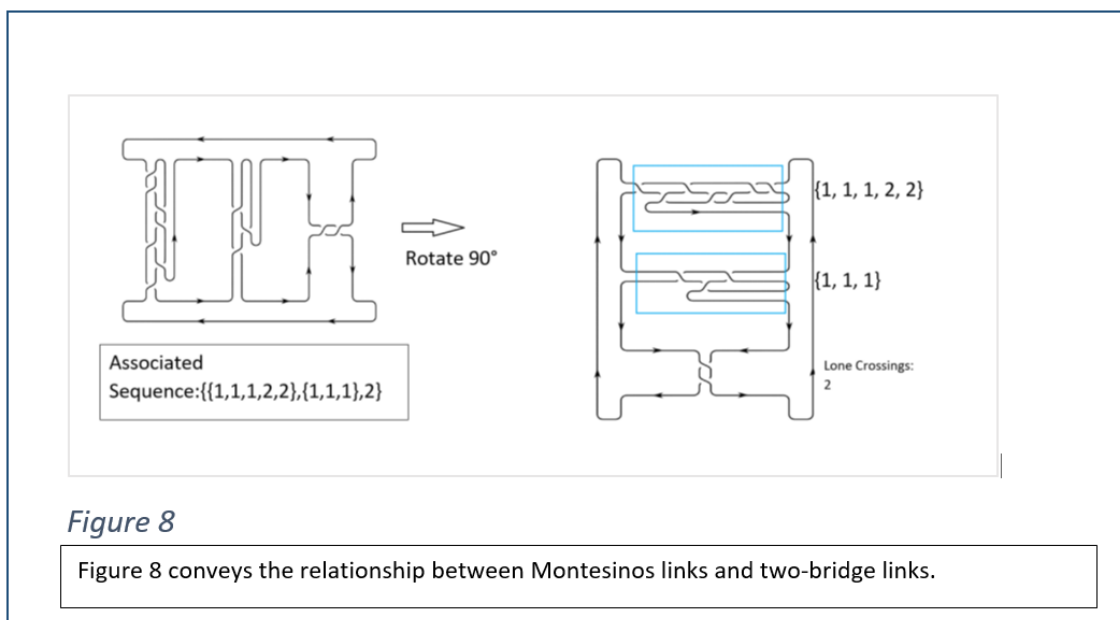
Construction of Montesinos Links

Alternating Montesinos links are represented by a sequence of continued fractions, as opposed to a single continued fraction as two-bridge links are represented. Each sequence represents an individual tangle in the knot diagram. For example, the sequence $(12/19, 2/3, 2)$, is a Montesinos Link of class M1. This is represented in continued fraction form as $\{1, 1, 1, 2, 2\}, \{1, 1, 1\}, 2$, where 2 is the number of lone crossings. Lone crossings occur at the end of the sequence and are represented by δ in Figure 13. They will be described in further detail throughout the construction description.

In our description of our first increment, we described the relationship between a fraction and its corresponding continued fraction form. This same logic is applicable to Montesinos links; however, the continued fraction form is based on the fraction's reciprocal. This can be seen in the first elements in both the sequence list $\{1, 1, 1, 2, 2\}, \{1, 1, 1\}, 2$ and the fraction list $(12/19, 2/3, 2)$. The reciprocal of $12/19$ is

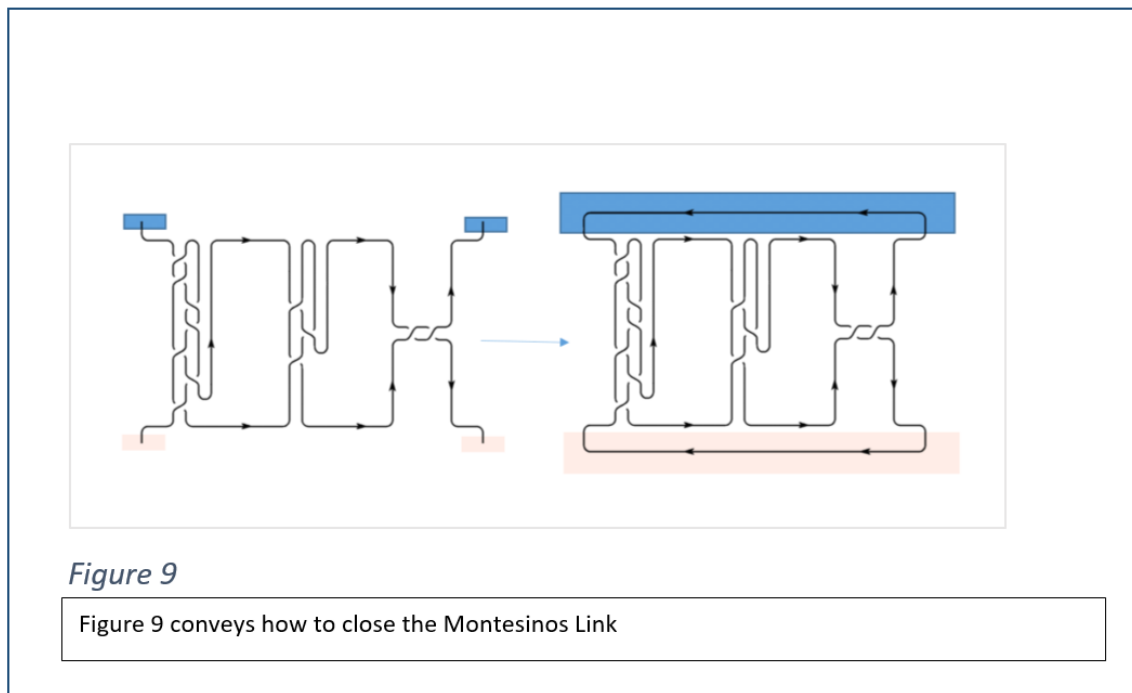
$$19/12, \text{ which is equivalent to } 1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{2 + \frac{1}{2}}}}.$$

Because Montesinos links consist of multiple continued fractions, there are structural comparisons that can be drawn between the two types of links. A Montesinos link can be visualized as a collection of two bridge links that are fused together. This is conveyed in Figure 8.



Because of this relationship, each integer sequence within the Montesinos sequence can be drawn as a two-bridge link, however, instead of closing the link at the end of its sequence, remaining strands to the right of the tangle are used to build the next two-bridge component specified in the sequence. Each

two-bridge component is an individual tangle within the link. Once every continued fraction sequence has been accounted for, reference the last element in the Montesinos sequence, which, unlike the other elements in the list, will be just an integer, as opposed to a list. This integer specifies the number of lone crossings which should be appended to the end of the link. To close the Montesinos link, connect the remaining strands on both the right and left side of the link as was done in the two-bridge link. We use the convention that the top strand in the link is oriented right-to-left after this closure. This is conveyed in Figure 9.



Defining Characteristics of M1, M2, and B Links

Our code functions for alternating links within classes M1, M2, and B. “Alternating” means that all strands in the diagram alternate between being over and under strands throughout the crossing interactions.

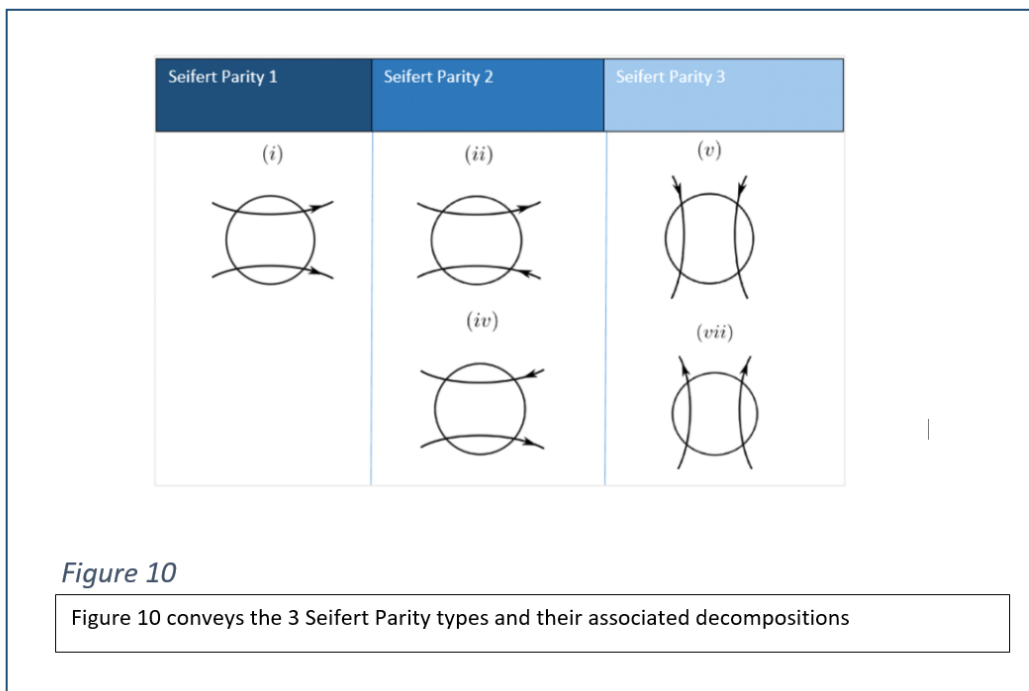
Montesinos links are categorized into the classes M1, M2, and B based on their Seifert parity decomposition. A Seifert decomposition is constructed by “smoothing” every crossing in the link diagram in the following way:



For each tangle, we examine the pair of arcs resulting from the Seifert decomposition which enter and exit the tangle. This pair of arcs, each of which use two of the four entering and exiting strands of the tangle, is associated with each tangle in a link diagram. The tangle's Seifert Parity is determined by how the arcs enter and exit the tangle.

There are three Seifert Parity types, conveyed in Figure 10.

Every tangle within Montesinos Links of class M1 decomposes as Seifert Parity 1, type (i). Every tangle in a Montesinos Link of class M2 decomposes as type (ii). B links contain at least one tangle that decomposes as one of the instances described in Seifert Parity 3.



For example, Figure 11 shows the Seifert parity decomposition of a link in class M1, $\{1,1,1,2,2\}, \{1,1,1\}, 2$. Figure 11 illustrates that following the orientations of the strands outlining the two sides of the tangles produces two parallel arcs that are both oriented towards the left, correspond-

ing to Seifert Parity

1.

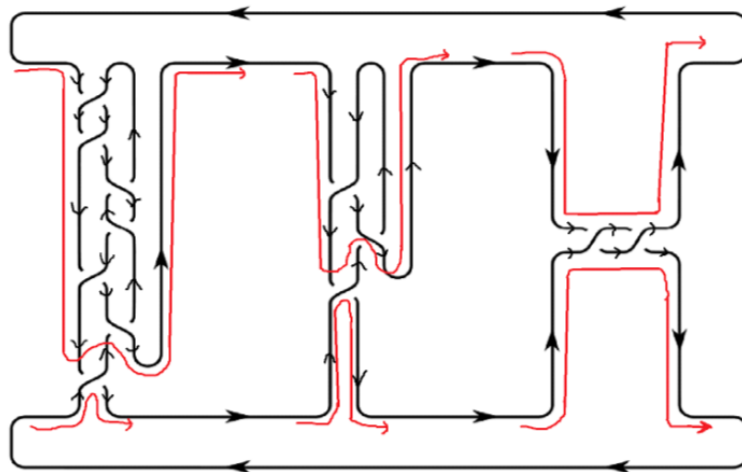


Figure 11

Figure 11 conveys the parallel arcs drawn for an M1 link.

An example of an M2 link Seifert parity decomposition is conveyed in Figure 12 . This figure illustrates that following the orientations of the strands outlining the two sides of the tangles produces two anti-parallel arcs, one pointing to the left, and the other to the right.

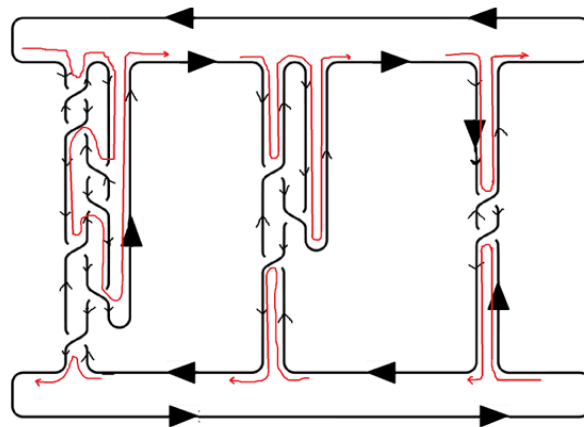


Figure 12

Figure 12 conveys the anti-parallel arcs drawn for a M2 link.

An additional distinguishing characteristic between links belonging to class M1 and class M2 is the orientation of the bottom long strand in both. In M2 cases, the bottom long strand is orientated from right to left, represented by “L”. This means the bottom long strand and the top long strand are anti-parallel in M2 cases. In M1 cases, the bottom long strand is oriented from left to right, represented by “R”. This means the bottom long strand and the top long strand are parallel.

Links within class B are further distinguished because only the top long strand has a set orientation, not the bottom long strand. The top long strand is expected to be oriented right to left, represented by “L”.

Furthermore, links of class M1 have lone crossings that are negative, links of class M2 possess no lone crossings, and links of class B possess positive lone crossings. Lone crossings are represented by the last integer in the inputted list and occur at δ in Figure 13.

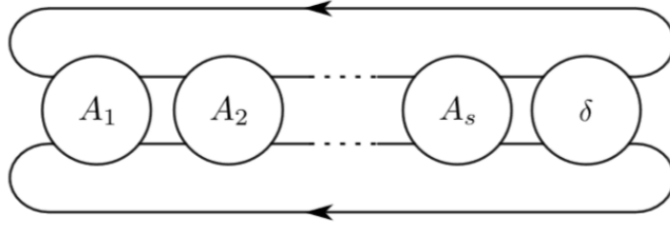


Figure 13

Figure 13 conveys the sequence of tangles that consist in both M1 and M2 links. The additional δ section represents lone crossings that can exist in links of class M1.

Associated Equations for M1, M2, and B Links

$$b(L) = 2 + \sum_{1 \leq j \leq k} \Delta_1(A_j) \text{ if } L \text{ is of Class M1}$$

The equation above is implemented within our code in the function “**braidIndexM**” and is applicable to two-bridge knots within the M1 class. 2 is added to the summation of each tangle implemented within its associated delta formula, $\Delta_1(A_j)$.

$$\Delta_1(A_j) = \frac{-1 + \text{sign}(b_{2,q_j+1}^j)}{4} + \sum_{b_{2,m}^j > 0, 1 \leq m \leq q_j} \frac{b_{2,m}^j}{2} + \sum_{b_{2,m+1}^j < 0, 0 \leq m \leq q_j} \frac{|b_{2,m+1}^j|}{2}$$

The above equation is the delta formula associated with M1 links and is implemented in the function **delta**. The operations performed in the delta are dependent on the signed vector of the specified tangle.

$$b(L) = 1 + \sum_{1 \leq j \leq k} \Delta_2(A_j) \text{ if } L \text{ is of Class M2}$$

The equation above is implemented within our code in the function “**braidIndexM**” and is applicable to two-bridge knots within the M2 class. 1 is added to the summation of each tangle implemented within its associated delta formula, $\Delta_2(A_j)$.

$$\Delta_2(A_j) = \frac{1 + \text{sign}(b_{2,q_j+1}^j)}{4} + \sum_{b_{2,m}^j > 0, 1 \leq m \leq q_j} \frac{b_{2,m}^j}{2} + \sum_{b_{2,m+1}^j < 0, 0 \leq m \leq q_j} \frac{|b_{2,m+1}^j|}{2}$$

The above equation is the delta formula associated with M2 links and is implemented in the function **deltas**. The operations performed in the delta are dependent on the signed vector of the specified tangle.

$$b(L) = \Delta_0(L) + \sum_{A_j \in \Omega_2} \Delta_2(A_j) + \sum_{A_j \in \Omega_3} \Delta_3(A_j) \text{ if } L \text{ is of Class B}$$

The equation above is implemented within our code in the function “**braidIndexBEquation**” and is applicable to two-bridge links within the B class. 1 is added to the summation of each tangle implemented within its associated delta formula, $\Delta_2(A_j)$.

$$\Delta_3(A_j) = \frac{-1 + \text{sign}(b_{2,q_j+1}^j)}{4} + \sum_{b_{2,m}^j > 0, 1 \leq m \leq q_j} \frac{b_{2,m}^j}{2} + \sum_{b_{2,m+1}^j < 0, 0 \leq m \leq q_j} \frac{|b_{2,m+1}^j|}{2}$$

The above equation is the delta formula associated with B links and is implemented in the function **deltas**. The operations performed in the delta are dependent on the signed vector of the specified tangle.

$$\Delta_0(L) = \eta + e - \min\{(\eta + e)/2 - 1, e\}, \text{ where } e \text{ is the number of lone crossings and } \eta = |\Omega_3|$$

The above equation is a delta formula associated with B links, implemented in the function **delta0**. The operations performed in this delta method use the number of Seifert Parity 3 tangles in the link, and the number of lone crossings in the link.

Increments Two and Three: Braid Index of Alternating Montesinos Links *Signed Vector*

Function Evaluation to Determine Signed Vectors of M1 and M2 Links

As seen in Figure 8, Montesinos links bear resemblance to two-bridge links, as they appear as a collection of two bridge links fused together with the potential addition of lone crossings. Consequently, our evaluation of Montesinos links parallels our evaluation of two-bridge links.

The calculation of the braid index is dependent on the signed vector for Montesinos links.

The signed vector corresponds to each twisting set within each tangle of the Montesinos link, meaning each integer in the given sequence of sequences. All of the crossings in a single twisting set have the same sign.

When given the signed vector that represents a continued fraction, the other functions needed to determine the braid index of a two-bridge link are **deltas** and **braidIndexM**. The function **braidIndexM** is a general function for the equation for the braid index of links that are either within class M1 or within class M2. The function **sumOfValuesAtIndices** is a helper method for this function, which returns the value of one of the two summations in the formula corresponding with the parity of an element's index. However, the given sequence is not signed, and thus the signed vector must be determined by additional functions.

Because of the close similarities between two-bridge links and Montesinos Links, we can evaluate the sign of the twisting sets within the links in the same way we would evaluate two-bridge links. The difference, however, is that we would evaluate this for multiple tangles which start at different orientations than how the two-bridge link started. However, each tangle within the same Montesinos link starts at the same orientation. The differences in our evaluation can be studied in the addition of functions **directionsM** and **toSignedVectorM**.

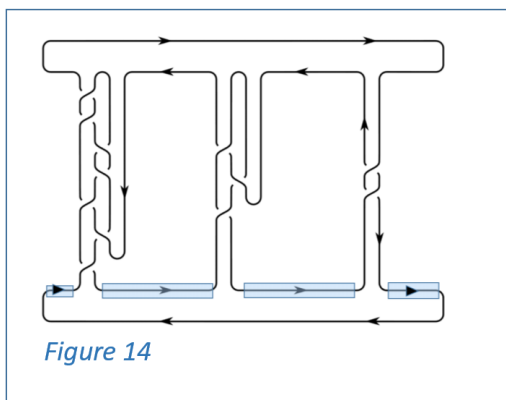
The function **directionsM** differs from **directions** in that it does not utilize the function **leftMostOrientations** to determine the leftMostOrientation of the Montesinos links, but instead determines this on the basis of which Montesinos class the link belongs to. Montesinos knots belonging to class M1 will have the leftmost orientation $\{\{R,R,L\}\}$ for each tangle, and Montesinos links belonging to class M2 will have the leftmost orientation $\{\{L,R,L\}\}$ for each tangle.

The function **toSignedVectorM** differs from **toSignedVector** because parallel orientations differ depending on the class of Montesinos link. In the case of class M1, the parallel orientations are $\{R,R\}$. In the case of M2, the parallel orientations are $\{L,L\}$. "Parallel orientations" refers to which orientations

the two strands involved in the crossing interaction will have if they are oriented in the same direction. This is significant information because it determines whether the twisting set is positive or negative.

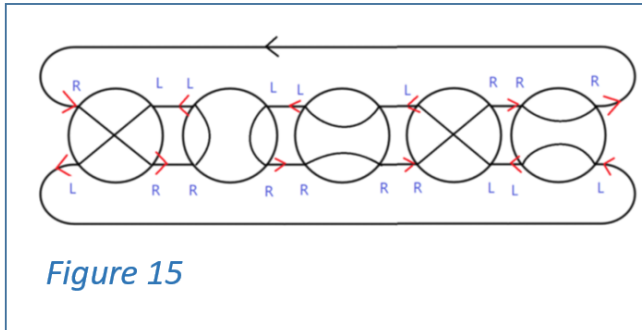
Determining the Signed Vector for B Links

Type B MontesinosLinks introduce an additional computational challenge because their tangles' leftmost orientations are not specified by convention as M1 and M2 links are: instead of the three leftmost orientations within each tangle being determined on the basis of class, the three leftmost orientations of each tangle are determined on the basis of the two bottom incoming strands' orientations. These incoming bottom strands are indicated in blue in Figure 14.

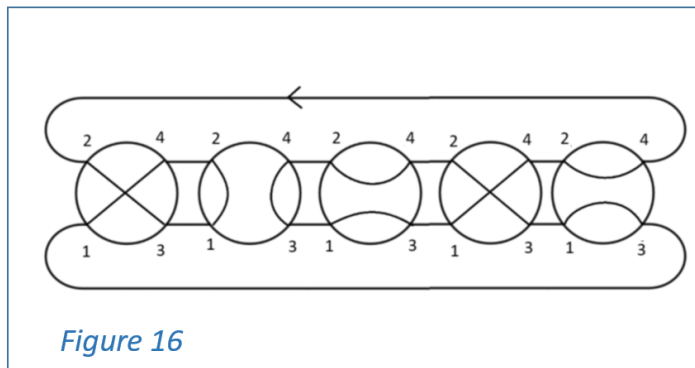


To determine the orientations of the two bottom incoming strands at each tangle, we needed to trace thorough the entire link to record the orientations of all strands entering each tangle. The figure below shows the four entering strands in each tangle.

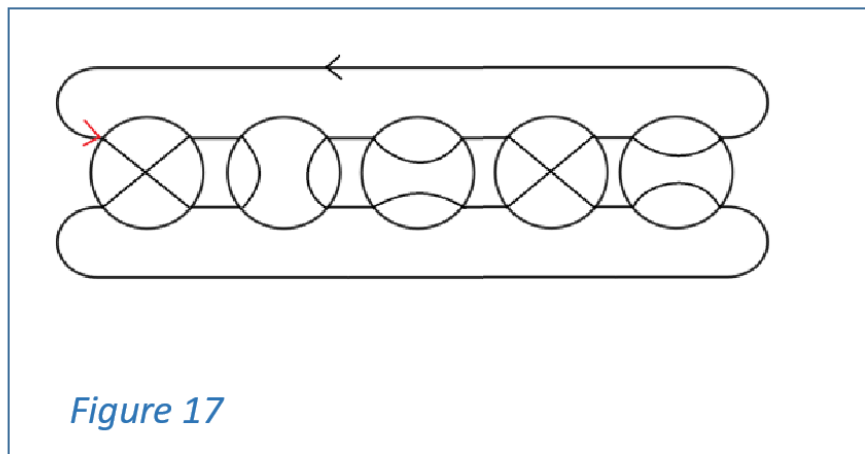
Our function **fourDirectionsOfmLink** accomplishes this by creating a list of the recorded orientations of each of the four strands at each tangle. For example, the sequence $\{\{“L”, “L”, “R”, “L”\}, \{“R”, “L”, “R”, “L”\}, \{“R”, “L”, “R”, “L”\}, \{“R”, “L”, “L”, “R”\}, \{“L”, “R”, “L”, “R”\}\}$ is associated with Figure 15. Each list of four directions corresponds to a tangle, and each letter corresponds to an orientation of one of the components.



We consider each of these four incoming strands as defined at 1 of 4 positions for each tangle. Each tangle possesses its own set of 4 positions. The labels of each incoming strand is in Figure 16. This also determines the order of directions given for each tangle.

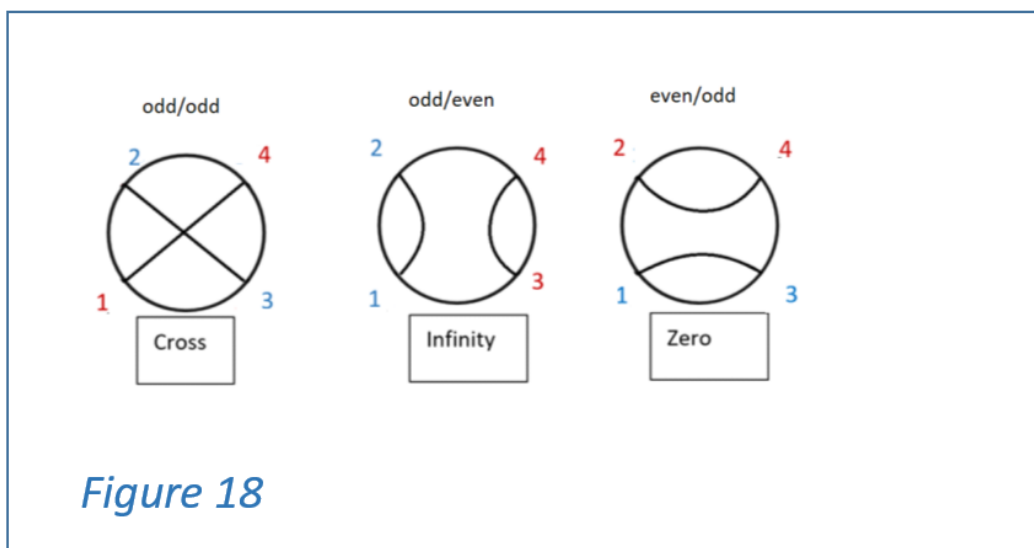


To trace around the link, we evaluate what we know about the link on the basis of convention. Convention indicates that the top strand in the link is oriented right to left. This means that when the strands turns 180 degrees to approach the first tangle, it is oriented in the opposite direction. This gives us the orientation "R" of incoming strand 1 in the left most tangle. When we begin, we start at tangle 1 at position 2. This is illustrated in Figure 16.

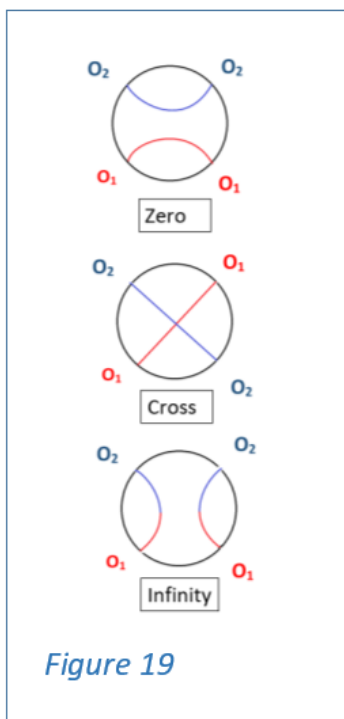


Once we established the position at which the strand enters the tangle, we then determine the new position the strand travels to in order to exit the tangle. We evaluate this using the function `switchPositionsInTangle`. This function determines the resulting exit position based on whether the tangle can be described as a cross tangle (note that this is commonly known as a Parity 1 tangle--we renamed this case in order to avoid confusion with the Seifert Parity 1), infinity tangle, or a zero tangle.

The different types of tangles are in the figure below. The method **`classifyTangle`** determines which type a tangle is using the tangle's fractional representation. If the fraction is of the form Odd/Odd, it is a Cross; if the fraction is of the form Odd/Even, it is Infinity; if the fraction is of the form Even/Odd, it is Zero. The tangle type indicates the expected exit position given the incoming position of the strand. The new position in a tangle can be determined by finding the same colored integer in each of the examples.

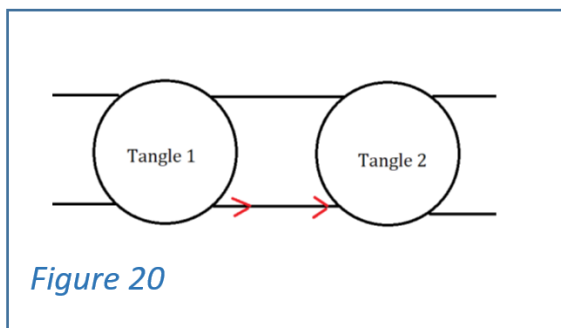


Next, our function **`switchDirections`** determines whether there is a directional change that occurs within the tangle. This would imply the incoming orientation of the strand that is being traced differs from the orientation at the exiting position. This is also determined by whether the tangle is a cross, infinity, or zero tangle. If the tangle is a cross or zero tangle, the orientation of the strand does not change within the tangle. If the tangle is an infinity tangle, the direction of the strand when it exits is opposite of what it was at its entrance. You can see this difference illustrated in Figure 19, where O_2 represent two different orientations. You can see whether the orientations are consistent or differ by comparing the orientations at either side of the tangle.

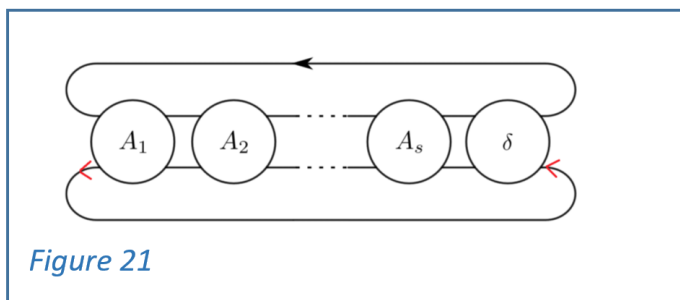


We then update the output four-directions list with the new direction and position for the current tangle.

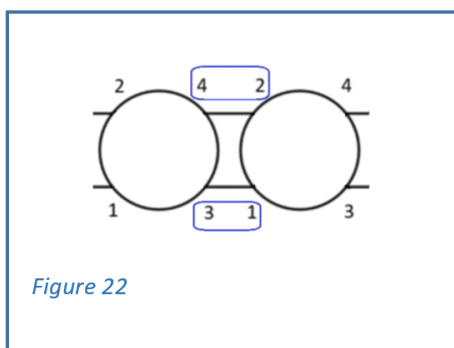
The next step is to continue following the strand out of the current tangle and into the next tangle. The strand entering the next tangle will have the same direction as it had when it exited the previous tangle. This can be clearly seen in the general example below, shown in Figure 20.



Note that even when we move from the first tangle to the last tangle along the strand spanning the bottom of the link, the orientation remains this same. This is because there is a 360 degree rotation.



We now must update the current position. Strands at position 4 in the current tangle transition to position 2 in the next tangle, and vice versa. Strands at position 1 in the current tangle transition to position 3 in the next tangle, and vice versa. We update both the current tangle and the position within the tangle.



We then have identified the location of the next tangle, the position of one of its incoming strands, and its orientation. Using these, we can repeat the process outlines above. This process continues until $4 \times (\text{length of the tangle list})$ labelings occur. (Note that if the link has more than one component, the four-directions list will not be complete. We use the function `fourDirectionsOfmLinkSecondComponent` to add an additional oriented component to the four-directions list. `fourDirectionsOfmLinkAllPossible` uses this function to return a list of all possible four-directions lists for a given Montesinos link.) This information then can be utilized to determine the leftmost orientations of the incoming strands of the link, and ultimately, determine the braid index, using the signed vectors calculated from this information.

4. File structure

We imported data from KnotInfo to test the `braidIndexTwoBridge` function.

5. Major data structures

Two-bridge links are represented by a list of integers which are in continued fraction form, meaning they are representative of a fraction. For example, the sequence representing the fraction $\frac{218}{101}$ is

$$\{2, 6, 3, 4, 1\}, \text{ since } \frac{218}{101} = 2 + \frac{1}{6 + \frac{1}{3 + \frac{1}{4 + \frac{1}{1}}}}. \text{ Note that } 4+1 \text{ could also be represented as } 5, \text{ producing}$$

an equivalent answer. However, the fraction is formatted in this way because the algorithms we use to determine the braid index of a two-bridge links must be of odd length. The odd-length sequence represents the two-bridge link which is built in the way described in Section 3.

A two-bridge link can also be represented as the single fraction formed from its sequence of integers. Under this convention, the two-bridge link described above would be represented $\frac{218}{101}$.

Montesinos links are represented as a sequence that begins with lists, each representing an individual tangle within the link, and ends with an integer that represent the number of lone crossings.

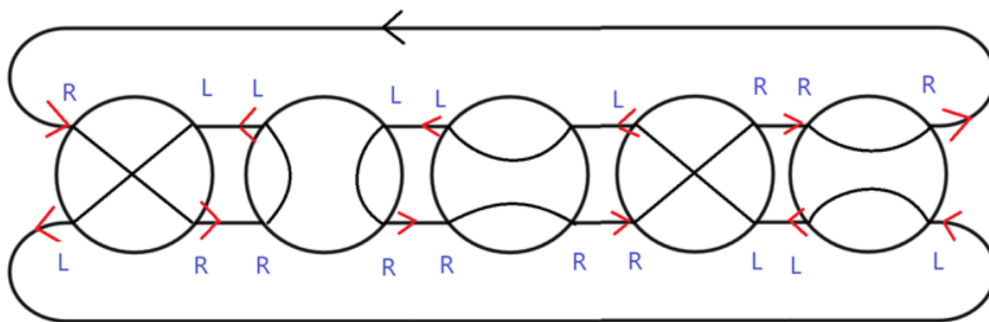
For example, one Montesinos Link is represented as $\{\{2, 2, 1, 1, 1\}, \{1, 1, 1\}, 2\}$. Each integer sequence represents a tangle within the link and the final integer represents the number of horizontal half-twists that occur at the end of the link. This is again described under Section 3 within our description of Montesinos link construction.

A Montesinos link can also be represented as a string in Conway notation. Conway notation is identical to the above representation except that: the tangle sequences are separated by semicolons, rather than being members of separate lists; each tangle is represented in a backwards order from the above representation; and the lone crossings are preceded by a “+” rather than a semicolon. For example, the above Montesinos link $\{\{2, 2, 1, 1, 1\}, \{1, 1, 1\}, 2\}$ would be represented as “1 1 1 2 2; 1 1 1 + 2”.

Montesinos links can also be represented as a list of their tangles’ fractions, with the number of lone crossings appearing at the end of the list. For example, the tangle $\{\{2, 2, 1, 1, 1\}, \{1, 1, 1\}, 2\}$ would be represented $\left\{\frac{19}{8}, \frac{3}{2}, 2\right\}$ in this way.

A **four-directions** list is a list of length-4 lists of strings which can be either “L” or “R”. Each of these lists represents the orientations of the four strands exiting and entering the relevant tangle. For example, the four-directions list $\{\{“L”, “L”, “R”, “L”\}, \{“R”, “L”, “R”, “L”\}, \{“R”, “L”, “R”, “L”\}, \{“R”, “L”, “L”, “L”\},$

“R”,{“L”, “R”, “L”, “R”}} is associated with the figure below:



The indices in a four-directions list are, clockwise starting from the upper left: 2,4,3,1.

6. Overview of the framework of the solution

The program calculates the braid index of two-bridge links and Montesinos links of type M1, M2, and B, using the formulas derived in “A Diagrammatic Approach for Determining the Braid Index of Alternating Links.”

Two-bridge Links

List of important functions:

-correctSequence[sequence_]: Given a sequence representing a continued fraction, outputs a sequence with an odd number of elements representing the same fraction.

-leftMostOrientations[sequence_]: Given a sequence of integers representing a link in two-bridge notation, determines whether the orientations of the strands preceding the leftmost set of crossings in the link are RLR or LRR by tracing the third strand around the knot/link. (Output is arbitrary if input is a link)

-directions[sequence_, choiceOfOrientation_]: Given a sequence representing a two-bridge link, and the choice of orientation of the leftmost set of strands, determines the direction of each set of three strands preceding each set of crossings, starting from the left and proceeding to the right.

-toSignedVector[sequence_, directionsChoice_]: Given a sequence representing a two-bridge link, and the orientations of each set of three strands preceding each set of crossings, outputs the signed vector for that two-bridge link.

- **sumOfValuesAtIndices[sequence_, parity_]**: computes one of the two summations in the braid index formula for two-bridge links given the sequence representing the link written in two-bridge notation. If the input parity is EvenQ, the summation involving the even indices in the sequence is computed; if the input parity is OddQ, the summation involving the odd indices in the sequence is computed.

- **braidIndexNumber[signedVector_]**: Given a signed vector, computes the braid index of the two-bridge link represented by the signed vector.

- **braidIndexTwoBridge[sequence_]**: Outputs the braid index (or a list of both possible braid indices, if the input is not a knot) of a two-bridge link represented by a sequence.

- **braidIndexForFraction[frac_]**: Outputs the braid index (or a list of both possible braid indices, if the input is not a knot) of a two-bridge link represented by a fraction.

How the functions are used:

Consider a sequence representing the continued fraction for the two-bridge link of interest, L. correctSequence converts this sequence to an equivalent sequence of odd length.

If L has only one component (i.e., is a knot), then there is only one possible orientation for its component, since our convention that the bottommost strand is oriented right-to-left would determine the orientation of the entire knot. leftmostOrientations determines the orientations of the three strands preceding the leftmost set of crossings of the two-bridge knot.

The directions method is given the sequence and a choice of starting orientations. For a knot, there is only one possible choice of starting orientations (these are determined by leftmostOrientations), and for a link, there are two possible choices of starting orientations (one for each of the two possible orientations of the upper component). The output is a list of the orientations of each of the three strands preceding each set of crossings, starting from the leftmost set and going right.

The toSignedVector method uses the output of the directions method to determine the sign of each set of crossings. The output is the signed vector for the oriented link.

braidIndexTwoNumber is given the signed vector and outputs the braid index using the sumOfValuesAtIndices method.

braidIndexTwoBridge computes all possible braid indices for L given its representation as a sequence.

If L is a link with two components, `braidIndexTwoBridge` returns a list with the outputs of `braidIndexTwoNumber` computed from both of the possible signed vectors for L . If L is a knot, `braidIndexTwoBridge` returns the only possible braid index.

`braidIndexForFrac` works in the same way as `braidIndexTwoBridge` except that its input is formatted as a fraction.

Montesinos Links

List of important functions:

-directionsM[sequence_,mVariety_]:

Input: sequence of numbers which correspond to the continued fraction representing a tangle, and the class (M1 or M2) of the link which the tangle is a part of

Output: List representing the directions of each of the three strands which precede each crossing set. This method returns the directions of the three strands to the left of each crossing set in the tangle, starting from the leftmost crossing set (which appears at the top when viewed as a part of a standard M link diagram) and proceeding to the right, when the tangle is drawn in an orientation where the crossing set corresponding to the first element in its sequence appears rightmost in the diagram.

-toSignedVectorM[sequence_, directionsChoice_,mVariety_]:

Input: sequence of integers representing a rational tangle, choice of orientations of strands preceding each set of crossings in the tangle, and the class of the link that the tangle is in ("M1" or "M2")

Output: the signed vector of the tangle.

-deltas[sequence_,seifertParityType_]: Given a tangle A represented by a signed vector and its Seifert parity p , where $p = 1$ or 2 , returns the value of the appropriate delta formula evaluated at that tangle; i.e. $\Delta_p(A)$.

-delta0[seifert3tangles_,loneCrossings_]: Given a list of the parity 3 tangles in a type B Montesinos link L , and the number of lone crossings in the link, outputs $\Delta_0(L)$ according to the formula in the paper.

-braidIndexM[mLink_,mType_]: Returns the braid index of a type M1 or type M2 link given a list `mLink` representing the input Montesinos link, and the type `mType` of the link. `mLink` is a list consisting of sequences (formatted as lists) representing the tangles in the link, ending with the number of lone

crossings (formatted as a single integer). The second parameter is a string, “M1” or “M2”.

- **braidIndexMForFraction[mLink_, mType_]**: Returns the braid index of a type M1 or type M2 link given a list mLink representing the input Montesinos link, and the type mType of the link. mLink is a list consisting of fractions (formatted such that numerator \leq denominator) representing the tangles in the link, ending with the number of lone crossings (formatted as a single integer). The second parameter is a string, “M1” or “M2”.

- **fourDirectionsOfmLink[mLink_]**: Returns a list of the lists of directions of four incoming strands of each tangle.

- **fourDirectionsOfmLinkSecondComponent[mLink_, fourDirections_, direction_]**: Returns a list of the lists of directions of four incoming strands of each tangle in mLink given the current directions list and the orientation of the bottom strand of the second component from the left.

- **braidIndexB[mLink_]**: Given a Montesinos knot of type B, outputs the braid index.

- **braidIndexBMultipleComponents[mLink_, fourDirections_]**: given a type B Montesinos link and its fourDirections list, outputs the braid index.

- **braidIndexBEquation[mLink_, twoIncomingOrientations_]**: outputs the braid index B calculation given the mLink and its list of 1 and 3 positions of its four directions list

- **tangleParity[sequence_]**: Given a sequence representing a tangle, outputs its Seifert Parity (1, 2, or 3).

- **seifertClass[mLink_]**: Given a Montesinos link, outputs its class (type M1, M2, or B).

- **braidIndexUnspecified[mLink_, fourDirections_]**: Given an oriented Montesinos link whose Seifert parity type is not given, along with its fourDirections sequence, outputs the braid index of the link.

- **braidIndexUnspecifiedForFraction[mLink_, fourDirections_]**: Given an oriented Montesinos link (written as a list of fractions representing rational tangles and an integer representing the number of lone crossings) whose Seifert parity type is not known or given, and its fourDirections list, outputs the braid index of the link.

How the functions are used:

Consider an M Link L (which is a type M1 or M2 link, or a type B knot) represented by a list of sequences, each representing a tangle in the link in left-to-right order, followed by an integer representing the

number of lone crossings. The function `braidIndexUnspecified` determines the braid index of L . (`braidIndexUnspecifiedForFraction` is the equivalent function for a Montesinos link represented in fraction form.)

First, the class of L must be determined. This is done by first determining the orientations of each set of four entering strands of each tangle using the `fourDirectionsOfmLink` function.

Then, the `tangleParity` method analyzes each of the sequences representing a tangle in L . `tangleParity` determines the Seifert parity type of a tangle (i, ii, iv, v, vi, vii). Based on the output of `tangleParity` of all the tangles in L , `seifertClass` returns the class (M1, M2, or B) of L : if all the tangle parities are 1 (i or ii), the link is of class M1. If all the tangle parities are 2 (iv or v), the link is of class M2. If the link has at least one tangle of parity 3 (vi or vii), the link is of class B.

Based on its class, L is input into either `braidIndexBMultipleComponents` (if it is of type B) or `braidIndexM` (if it is of type M1 or M2). These braid index calculating methods utilize the `deltas` and `delta0` methods to compute the summations in the appropriate braid index formula.

The `deltas` and `delta0` methods use the signed vector of a tangle given its two-bridge sequence for their computations. The signed vector representing a tangle is determined by the methods `directionsM` and `toSignedVectorM`. The `directionsM` method determines the directions of the sets of three strands preceding each set of crossings in the tangle, and `toSignedVectorM` determines the signed vector for the tangle using these directions, similar to the `directions` and `toSignedVector` methods from the two-bridge case.

`braidIndexBMultipleComponents` takes the sum of the `deltas` method evaluated for each of the tangles according to their Seifert parity. For example, if tangle T was of parity 3, `deltas[sequenceT, 3]` would be added to the braid index calculation, where `sequenceT` is the sequence of integers representing T . The method `delta0` evaluated for L is added to this sum for the final result of the braid index calculation.

For M1 links, `braidIndexM` computes the sum of the Δ_1 formula evaluated at each tangle and adds 2 to the result. For M2 links, `braidIndexM` computes the sum of the Δ_2 formula evaluated at each tangle and adds 1 to the result.

The braid index calculation result from the appropriate method is returned.

For a type B link with multiple components, the `fourDirectionsOfmLinkSecondComponent` is used multiple times to determine the orientations in `fourDirections`, based on the specified orientation of the component. This list of four directions is what is input into the braid index B calculation in the `braidIndexBMultipleComponents` function.

7. Externally used code

All the code in our project is our own.

8. Limitations

Our code has not been able to compute the braid index of links with over 29 tangles after 20 minutes. It can generally compute the braid index of links with up to 10 tangles with reasonably instantaneous speed; at over 15 tangles, the speed is much less reliable.

9. References

Diao, Yuanan, Ernst, Claus, Heteyi, Gabor, & Liu, Pengyu. "A Diagrammatic Approach for Determining the Braid Index of Alternating Links." (2018 pre-print.)

KnotInfo Table of Knot Invariants (<https://www.indiana.edu/~knotinfo/>).