ThreeBraidClass　モジュール　利用の手引き

[概要]

ThreeBraidClass モジュールは、3-braid の様々な表示形式とそれらの間の変換や、3-braid の視覚化のための python モジュールです。３つのクラスと、3-braid の表示形式の間の変換関数、それらを扱うための関数などが定義されています。３つのクラスは、様々な表示形式を内包し一般的な 3-braid を扱うための **ThreeBraid** クラス、論文 [Og] をもとに linear 3-braid（Lissajous 3-braid を含む）を扱うための **LinearBraid** クラス、論文 [KN01]/[KN02] に基づき Lissajous 3-braid を扱うための **LissajousBraid** クラスです。**ThreeBraidClass** モジュールの利用のために他のモジュールを明示的に組み込む必要はありませんが、python に標準モジュールとして用意されている **math**, **cmath**, **fractions** を組み込むこと（**import**）を推奨します。また、様々な表現形式で表された 3-braid の視覚化のためには、**numpy**, **matplotlib**, **ipywidgets**, **IPython** モジュールがインストールされ、**jupyter** などのグラフ表示可能な端末上で実行される必要があります。

[準備]

python 標準モジュール **math**, **cmath**, **fractions** と、**ThreeBraidClass** モジュールを組み込んでください。ただし、最初の３つの標準モジュールは組み込まなくても動作します。

```
>>> import math          \# 標準的な関数
>>> import cmath         \# 標準的な複素関数
>>> from fractions import Fraction      \# 分数環境
```

ThreeBraidClass モジュールを組み込む。

```
>>> import ThreeBraidClass as tb        \# 3-braid 計算環境
```

python では組み込んだモジュールで定義された関数などを利用する際に、頭にモジュール名をつけます。**ThreeBraidClass** モジュールの名称は少し長いので省略形 **tb** で利用できるように **import** 文に **as tb** をつけています。

```
########################################
## ThreeBraidClass module              ##
##      successfully imported          ##
##      ver.1.2.4  2022/10/24  H.Ogawa ##
########################################
```

[ThreeBraid クラス]

ThreeBraid クラスは、さまざまな形で表された 3-braid （upto full-twist） について、別の形での表示や、簡約形を得るための計算手続きをまとめたものです。使利用手順は、**[インスタンスの生成]**、**[3-braid 入力]**、**[結果出力]** となります。また、**[3-braid の連結（加法演算）]**や **[3-braid の比較]** もできます。

[インスタンスの生成]　　インスタンスとは、クラス型オブジェクトのこのとです。**ThreeBraid** クラスを利用するために、**ThreeBraid** クラス型オブジェクトを参照する変数、例えば **br** を用意します。これをインスタンスの生成と言います。変数の名称は任意です。

```
>>> br = tb.ThreeBraid()
```

tb は **ThreeBraidClass** モジュールの略称で、**ThreeBraidClass** モジュールの中の **ThreeBraid** クラス定義（**tb.ThreeBraid()** のこと）を呼び出して **ThreeBraid** クラス型オブジェクトを生成し、変数 **br** に割り当てると言う意味です。後述しますが、**ThreeBraid** クラスに引数を指定することもできます。引数を指定しすることで、**[インスタンスの生成]** と次の **[3-braid 入力]** を同時に行えます。

[3-braid 入力]　**ThreeBraid** クラス型オブジェクト変数 **br** への 3-braid の入力は、メソッド **set()** を使います。クラス型オブジェクト変数へメソッドを適用ためには、変数にメソッドをドット **"."** でつなぎます。つまり、**br.set()** と記述します。**set()** メソッドの引数には、3-braid の表現形式を表す文字列またはリストを与えます。3-braid を特定するための表現形式には **AB**、**Sigma**、**BDPQ**(or **Frieze**)、**Syzygy**、**RL**、**ST**、**PSL2** の７つを採用しています。**PSL2** 型表現形式は行列（長さ２のリストを成分とする長さ２のリスト））で与え、他は文字列（'' もしくは "" で括られた文字の並び）で与えます。入力された文字列がどの表

現形式であるか、自動的に判別します。次の例は、**AB** 型表現形式で与えたものです。入力が正常に終了したとき、**The process completed successfully.** と表示されます。

```
>>> br.set('BBABBBBAAABBABABB')
```

計算終了のメッセージ出力を必要としないとき、**set()** メソッドのキーワード引数 **silinet** に真偽値 **True** を指定してください。

```
>>> br.set('BBABBBBAAABBABABB', silent=True)
```

ThreeBraid クラスで扱う 3-braid は、上端下端のそれぞれが一直線上に並んだものを想定しています。上端下端を並びの順につないで得られる閉じた 3-braid、同じ 3-braid を幾つかつないで shape sphere 上の閉路としたものも、同時に扱えるようにしています。**ThreeBraid** クラスでは、上端下端のあるものを **braid**、上端下端をつないだものを **closure**、shape sphere の閉路に対応するように伸ばして上端下端をつないだ **loop** の３種を **braid style** と呼び、**ThreeBraid** クラスの内部に、同じ名称のインスタンス変数をもち、それぞれの形に応じた７つの表現形式における簡約形などが格納されます。**set()** メソッドでは、これらのことが全て自動的に実行されます。**closure** は、表現形式を巡回語（circular word）とみること、つまり、braid 群における共役類を考えたものにあたります。**loop**は、与えられた 3-braid を高々３つつなげることで、shape sphere 上の対応する道が閉路になるようにしたもので、表現形式の文字列を高々３つつないが語の属する巡回語で与えられます。3-braid は上端３点から下端３点への置換を引き起こすので、高々３つつなげば、下端の点の配置が上端の点の配置にもどります。3-braid を水平方向の面で切った断面における３点の配置が、頂点の順序を指定した相似三角形の moduli 空間の shape sphere 上の点に対応するので、切る面を上端から下端に連続的に動かして、shape sphere 上の道が定まります。上端と下端の端点の配置を対応する点の順序も含めて同じになるようにしたので、その道は閉路になります。**braid** の表す 3-braid は shape sphere 上の道に対応しますが、**loop** の表す shape sphere 上の閉路の一部になっています。**braid** の上端下端における端点は一直線上にあるとしましたので、**ThreeBraidClass** モジュールでは、Shape sphere 上の道は基本的に、赤道（朔 syzygy の状態）上のある点を始点とし、赤道上の（別の）点を終点と考えます。表現形式によっては、端点の並び順などを指定する必要があります。これは、shape sphere 上の道の始点を指定することに対応します。始点の指定にはキーワード引数 **startArea**を使います。shape sphere の赤道上の３点 １, ω, ω^2（ωは１の原始３乗根）は３本の紐のうち２本が交わる（collision）状態をを意味するので、道（path）はその３点を通りません。道の始点、終点として、その３点で切り分けられた赤道上の３つの区画（syzygy の状態）を考えればよい。また、道が赤道上を通り抜ける向き（３点が動き始めた瞬間における３点の配置の向き）も考えれば、**startArea** には、区画を表す三つの文字（0,1,2）に向きを表す符号（+ ,-）を添えた六種のうちの一つを指定することになります。**set()** メソッドで実行後、**braid**、**closure**、**loop** の３種それぞれに対しする７つの表現形式が、自動的に計算され **ThreeBraid** クラス型オブジェクト変数の内部インスタンス変数に格納されます。

ThreeBraid クラス型オブジェクト変数を生成する際に、**set()** メソッドに受け渡す引数を指定し、**[インスタンスの生成]** と **[3-braid の入力]** を同時に行うことができます。ただし、このとき silent 引数のデフォルト値は True に設定されています。　>>> **br = tb.ThreeBraid()**

```
>>> br.set('BBABBBBAAABBABABB', silent=True)
```

と

```
>>> br = tb.ThreeBraid('BBABBBBAAABBABABB')
```

は、同じ意味になります。後述しますが、後者を利用することで、変数に割り当てずに **ThreeBraid** クラスを利用することもできます。

[出力]　**ThreeBraid** クラス型オブジェクト変数 **br** に **set()** メソッドで 3-braid を入力すると、**braid**、**closure**、**loop** の３種それぞれに対して、前述の７つの表現形式と dilatation などが計算され、変数 **br** の内部インスタンス変数 **braid**、**closure**、**loop** に格納されます。 **br.braid** などとして計算結果を参照することもできますが、モジュール内での計算に使われている情報なども表示されるので、やや冗長なものになります。内部インスタンス変数は、python における辞書型オブジェクトなので、**br.braid['AB']** のように、必要な情報を直接引き出すこともできます。一般的な用途における計算結果の参照には、**show()** メソッドを利用してください。

```
>>> br.show()
```

実際に計算してみます。AB 型表現形式 **'BBABBBBAAABBABABB'** で与えられた 3-braid を入力し、**show()** メソッドで結果を出力します。

```
>>> br.set('BBABBBBAAABBABABB')
```

```
The process completed successfully.
```

```
>>> br.show()
```

```
Braid Style: braid
  AB: BBABABBABABB
  Sigma: σ2^-1 σ1^-2 σ2 σ1^-1 σ2
  Syzygy: 12012
  BDPQ (Frieze): bqbqb
  RL: R^(-1) L^2 R L R
  ST: ST2 ST3 ST2
  PSL2: [[3, 5], [-5, -8]]
  Dilatation: (5 + sqrt(21)) / 2  (value=4.7912878474779195)
Braid Style: circular word
  AB: ABABBABAB
  Sigma: σ1^-1 σ2 σ1^-2
  Syzygy: 1201
  BDPQ (Frieze): qbqp
  RL: L R L^2
  ST: ST3 ST2 ST2
  PSL2: [[3, 1], [5, 2]]
  Dilatation: (5 + sqrt(21)) / 2  (value=4.7912878474779195)
Braid Style: loop on shape sphere
  AB: ABABBABABABABBABABABABBABAB
  Sigma: σ1^-1 σ2 σ1^-3 σ2 σ1^-3 σ2 σ1^-2
  Syzygy: 120101202012
  BDPQ (Frieze): qbqpqbqpqbqp
  RL: L R L^3 R L^3 R L^2
  ST: ST3 ST2 ST2 ST3 ST2 ST2 ST3 ST2 ST2
  PSL2: [[67, 24], [120, 43]]
  Dilatation: (110 + sqrt(12096)) / 2  (value=109.99090833947008)
Braid Style: loop/3 on shape sphere
  AB: ABABBABAB
  Sigma: σ1^-1 σ2 σ1^-2
  Syzygy: 1201
  BDPQ (Frieze): qbqp
  RL: L R L^2
  ST: ST3 ST2 ST2
  PSL2: [[3, 1], [5, 2]]
  Dilatation: (5 + sqrt(21)) / 2  (value=4.7912878474779195)
```

このように **show()** メソッドで、すべての計算結果が出力されます。**braid**、**closure**、**loop** の３種に対して、７つの表現形式での表示と dilatation が出力されます。、また、**loop** が同じ 3-braid ３つに分割できるときは **loop** の 1/3 長（**loop3**）についても計算され、出力されます。さらにもう半分（同一ではなく鏡映に場合も含めて）に分けられるときは 1/6 長（**loop6**）についても計算され、出力されます。**show()** メソッドに引数を与えて、必要なものだけを出力させることができます。例えば **closure** に関する情報のみを出力したい場合は、第１引数に **'closure'** と書く方法と、キーワード引数（**brStyle**）に **'closure'** を割り当てる方法があります。

```
>>> br.show('closure')
```

```
>>> br.show(brStyle='closure')
```

```
Braid Style: circular word
  AB: ABABBABAB
  Sigma: σ1^-1 σ2 σ1^-2
  Syzygy: 1201
  BDPQ (Frieze): qbqp
  RL: L R L^2
  ST: ST3 ST2 ST2
  PSL2: [[3, 1], [5, 2]]
  Dilatation: (5 + sqrt(21)) / 2  (value=4.7912878474779195)
```

show() メソッドのキーワード引数（**brStyle**）には、**'braid', 'closure', 'loop', 'loop3', 'loop6'** と、これらの任意の組（**list [], tuple ()**）を指定できます。すべてを指定する代わりに **'all'** や **''** を使うことができます。**brStyle** 引数の記述を省略した場合は **brStyle** に **'all'** が割り当てられます。

特定の表現形式のみを表示させたい場合は、**show()** メソッドの第２引数、あるいはキーワード引数 （**brType**） を使います。**brType** 引数には、７つの表現形式の他に、内部インスタンス変数の辞書キー（**keys()** で調べられます）を指定できます。**brStyle** と同様に組にして複数を指定することもできます。

```
>>> br.show(brType=['AB', 'Syzygy'])
```

```
Braid Style: braid
  AB: BBABABBABABB
  Syzygy: 12012
Braid Style: circular word
  AB: ABABBABAB
  Syzygy: 1201
Braid Style: loop on shape sphere
  AB: ABABBABABABABBABABABABBABAB
  Syzygy: 120101202012
Braid Style: loop/3 on shape sphere
  AB: ABABBABAB
  Syzygy: 1201
```

この例では **brStyle** 引数が省略されているので、**brStyle='all'** が指定されたことになります。キーワード引数は、値を割り当てる変数名が書かれているので、任意の順序で引数を記述できます。**brStyle** 引数と **brType** 引数の順序はどちらが先でもかまいません。引数名 **brStyle, brType** を省略する場合は、**brStyle**に割り当てる値を第１引数に、**brType** に割り当てる値を第２引数に書かなければなりません。

```
>>> br.show(brStyle=['braid', 'closure', 'loop'], brType=['AB', 'Syzygy'])
```

```
Braid Style: braid
  AB: BBABABBABABB
  Syzygy: 12012
Braid Style: circular word
  AB: ABABBABAB
  Syzygy: 1201
Braid Style: loop on shape sphere
  AB: ABABBABABABABBABABABABBABAB
  Syzygy: 120101202012
```

上端下端のある **braid** の表示式は始点終点のある文字列で表します。**closure**、**loop**、**loop3**、**loop6** の表示式は巡回語（circular words）として扱われます。**braid** は **set()** メソッドで与えられた文字列をその

まま用います。**closure** は、与えられた文字列を巡回語として扱います。**loop** は、shape sphere 上の閉路に対応するように、与えられた文字列を２つか３つつないだものを巡回語にして扱います。braid に対応する shape sphere 上の道を、次のように作ることができます。１、ω、ω^2と北極、南極を結ぶ子午線と赤道で、shape sphere を６つの領域に分割します。６つの領域の任意の一つの中の点を始点とし、braid の文字列（具体的には AB 型表現形式での A や B）に従って、南北や東西の隣の領域に移動する道が braid の対応する道（の属するホモトピー類の代表）となります。その道の始点と終点はそれぞれ6つの領域のいずれかに属します。**ThreeBraidClass** モジュールでは、始点と終点を赤道上の３つの区画のうちのひとつにとりその区画を表す文字(0,1,2) と、始点が北極側にあるときは **+** 、南極側にあるときは **-**を添えた、６種で表せます。Shape sphere 上の閉路になるためには、始点と終点の属する領域が同じでなければなりません。**loop** は、同じ 3-braid を何回か繰り返して、始点終点が赤道上の同じ区画の同じ向きになるようにしたものです。例えば、始点が **'0+ '** で、終点が **'1-'** の 3-braid を２つつないだとき、２つめの 3-braid は始点が **'1-'** になりますが、南半球で道の進む方角（東西方向）は、北半球で進む方向と逆向きになるので、最初の道と赤道に対して対称な道をたどることになり、終点が **'0+ '** になります。始点終点が赤道上の同じ区画で同じ向きになったので、**loop** は、3-braid を２倍に伸ばして、始点終点をつないだ circular word とみたものになります。**loop** にするためにくり返した回数は **show()** メソッドの引数 **brType** に **'multiple'** を指定することでわかります。**'multiple'** は **loop** に対してのみ意味があり、**braid**, **closure**, **loop3**, **loop6** では常に１です。

```
>>> br.show(brType='multiple')
```

```
Braid Style: braid
  multiple: 1
Braid Style: circular word
  multiple: 1
Braid Style: loop on shape sphere
  multiple: 3
Braid Style: loop/3 on shape sphere
```

ThreeBraid クラス型オブジェクト生成時に適当な表現形式を与え、続けて **show()** メソッドを適用し、結果を表示させることができます。変数に割り当てずに **ThreeBraid** クラスに組み込まれた計算アルゴリズムを利用できます。

```
>>> tb.ThreeBraid('BBABBBBAAABBABABB').show()
```

```
Braid Style: braid
  AB: BBABABBABABB
  Sigma: σ2^-1 σ1^-2 σ2 σ1^-1 σ2
  Syzygy: 12012
  BDPQ (Frieze): bqbqb
  RL: R^(-1) L^2 R L R
  ST: ST2 ST3 ST2
  PSL2: [[3, 5], [-5, -8]]
  Dilatation: (5 + sqrt(21)) / 2  (value=4.7912878474779195)
Braid Style: circular word
  AB: ABABBABAB
  Sigma: σ1^-1 σ2 σ1^-2
  Syzygy: 1201
  BDPQ (Frieze): qbqp
  RL: L R L^2
  ST: ST3 ST2 ST2
  PSL2: [[3, 1], [5, 2]]
  Dilatation: (5 + sqrt(21)) / 2  (value=4.7912878474779195)
Braid Style: loop on shape sphere
  AB: ABABBABABABABBABABABABBABAB
  Sigma: σ1^-1 σ2 σ1^-3 σ2 σ1^-3 σ2 σ1^-2
  Syzygy: 120101202012
  BDPQ (Frieze): qbqpqbqpqbqp
  RL: L R L^3 R L^3 R L^2
  ST: ST3 ST2 ST2 ST3 ST2 ST2 ST3 ST2 ST2
  PSL2: [[67, 24], [120, 43]]
```

```
  Dilatation: (110 + sqrt(12096)) / 2  (value=109.99090833947008)
Braid Style: loop/3 on shape sphere
  AB: ABABBABAB
  Sigma: σ1^-1 σ2 σ1^-2
  Syzygy: 1201
  BDPQ (Frieze): qbqp
  RL: L R L^2
  ST: ST3 ST2 ST2
  PSL2: [[3, 1], [5, 2]]
  Dilatation: (5 + sqrt(21)) / 2  (value=4.7912878474779195)
```

より電卓的（?）に、必要な答えだけ引き出すには、

```
        >>> tb.ThreeBraid('BBABBBBAAABBABABB').show('', 'Syzygy')


Braid Style: braid
  Syzygy: 12012
Braid Style: circular word
  Syzygy: 1201
Braid Style: loop on shape sphere
  Syzygy: 120101202012
Braid Style: loop/3 on shape sphere
  Syzygy: 1201
```

[使用例] **ThreeBraid** クラス型オブジェクト変数に再度 **set()** メソッドを呼び出すことで、別の 3-braid を入力できます。新たに ThreeBraid クラス型オブジェクトを生成し別の変数に割り当てることもできます。上と同じ **br** に、Sigma 表現形式で 3-braid を与えてみます。Sigma 表現形式は σ1、σ2 のべき乗の積の形で与えるのですが、その形の代用として、σ1 を **'1'** に、σ2 を **'2'** に、σ1^(-1) を **'-1'** に、σ2^(-1) を **'-2'** に置き換え、空白で区切り、冪指数の分だけ並べた文字列で表します。例えば、σ1^(-3) σ2^2 σ1^(-1) σ2^(-2) σ1 は、**'-1 -1 -1 2 2 -1 -2 1'** で与えます。

```
        >>> br.set('-1 -1 -1 2 2 -1 -2 1')


The process completed successfully.
```

```
        >>> br.show()


Braid Style: braid
  AB: ABABABABBABBABBABBA
  Sigma: σ1^-3 σ2^3 σ1^-1 σ2^-1
  Syzygy: 010121211
  BDPQ (Frieze): qpqbdbd
  RL: L^3 R^3 L R^(-1)
  ST: T ST2 ST2 ST5 S
  PSL2: [[4, -1], [13, -3]]
  Dilatation: (1 + sqrt(-3)) / 2  (value=(0.5+0.8660254037844386j))
Braid Style: circular word
  AB: BB
  Sigma: σ2^-1 σ1^-1
  Syzygy: ''
  BDPQ (Frieze): b
  RL: R^(-1) L
  ST: ST
  PSL2: [[0, 1], [-1, -1]]
  Dilatation: (1 + sqrt(-3)) / 2  (value=(0.5+0.8660254037844386j))
Braid Style: loop on shape sphere
  AB: ''
  Sigma: ''
  Syzygy: ''
```

```
  BDPQ (Frieze): ''
  RL: ''
  ST: ''
  PSL2: [[1, 0], [0, 1]]
  Dilatation: (2 + sqrt(0)) / 2  (value=1.0)
Braid Style: loop/3 on shape sphere
  AB: ''
  Sigma: ''
  Syzygy: ''
  BDPQ (Frieze): ''
  RL: ''
  ST: ''
  PSL2: [[1, 0], [0, 1]]
  Dilatation: (2 + sqrt(0)) / 2  (value=1.0)
Braid Style: loop/6 on shape sphere
  AB: ''
  Sigma: ''
  Syzygy: ''
  BDPQ (Frieze): ''
  RL: ''
  ST: ''
  PSL2: [[1, 0], [0, 1]]
```

上の計算結果を見ると、**braid** の表現形式には何らかの文字列が表示されていますが、**loop** などでは空の文字列（''）になっています。その理由を調べるために、計算の最初の素データ（raw data）を表示させてみます。**ThreeBraid** クラスでは、それぞれの **braid style** ごとに表現形式 **braid type** を計算していますが、そのための最初のデータが **raw** に格納されています。

```
>>> br.show(brType=['raw', 'AB', 'multiple'])
```

```
Braid Style: braid
  raw data:  -1 -1 -1 2 2 -1 -2 1
  AB: ABABABABBABBABBABBA
  multiple: 1
Braid Style: circular word
  raw data:  -1 -1 -1 2 2 -1 -2 1
  AB: BB
  multiple: 1
Braid Style: loop on shape sphere
  raw data:  -1 -1 -1 2 2 -1 -2 1  -1 -1 -1 2 2 -1 -2 1  -1 -1 -1 2 2 -1 -2 1
  AB: ''
  multiple: 3
Braid Style: loop/3 on shape sphere
  raw data: ''
  AB: ''
Braid Style: loop/6 on shape sphere
  raw data: ''
  AB: ''
```

circular word をとった **closure** の AB 型表現形式は **'BB'** です。従って、与えられた 3-braid を３つつなぐことで shape sphere の閉路になります。**loop** は、Sigma 表現形式で与えた文字列を３つつないだものが **raw** データとなり、これを AB 型表現形式に変換すると ''（自明な 3-braid）になります。これは **closure** の AB 型表現形式を３倍にのばした **'BBBBBB' = ''** とも矛盾しません。**loop3** の有無は、自明になった **loop** の AB 型表現形式が３分割できるかどうかで判断され、AB 型表現形式を３分割したものを **'raw'** データとしています。**loop6** は、**loop3** の AB 型表現形式をもとにしているので、**'raw'** データは **loop3** の AB 型表現形式の半分になります。以上のことから **loop**, **loop3**, **loop6** のほとんどの項目が '' になったのです。

Frieze pattern（BDPQ word）は4つの文字 p, b, q, d からなる文字列で、Syzygy 列は3つの文字 0, 1, 2 からなる文字列で与えます。RL 表現形式は R, L と r, l からなる文字列で、小文字 r, l はそれぞ

れ R と L の inverse を表します。これらの表現形式では、空白などの区切り文字は入れません。それぞれについて、'AB' 型表現形式に変換してみましょう。

```
        >>> br.set('pqdpdb', silent=True)


        >>> br.show(brType=['raw', 'BDPQ', 'AB'])


Braid Style: braid
  raw data: pqdpdb
  BDPQ (Frieze): bdb
  AB: BBABBABB
Braid Style: circular word
  raw data: pqdpdb
  BDPQ (Frieze): dp
  AB: ABBAB
Braid Style: loop on shape sphere
  raw data: pqdpdbpqdpdbpqdpdb
  BDPQ (Frieze): dpdpdp
  AB: ABBABABBABABBAB
Braid Style: loop/3 on shape sphere
  raw data: ABBAB
  BDPQ (Frieze): dp
  AB: ABBAB
Braid Style: loop/6 on shape sphere
  raw data: ABB
  BDPQ (Frieze): d
  AB: ABB


        >>> br.set('000112220102011', silent=True)


        >>> br.show(brType=['raw', 'Syzygy', 'AB'])


Braid Style: braid
  raw data: 000112220102011
  Syzygy: 020102011
  AB: ABBABBABABABBABBABA
Braid Style: circular word
  raw data: 000112220102011
  Syzygy: 0102
  AB: ABABABBAB
Braid Style: loop on shape sphere
  raw data: 000112220102011000112220102011
  Syzygy: ''
  AB: ''
Braid Style: loop/3 on shape sphere
  raw data: ''
  Syzygy: ''
  AB: ''
Braid Style: loop/6 on shape sphere
  raw data: ''
  Syzygy: ''
  AB: ''


        >>> br.set('rLLRLrLRlrL', silent=True)


        >>> br.show(brType=['raw', 'RL', 'AB'])
```

```
Braid Style: braid
  raw data: rLLRLrLRlrL
  RL: L^(-1)
  AB: BBA
Braid Style: circular word
  raw data: rLLRLrLRlrL
  RL: R
  AB: ABB
Braid Style: loop on shape sphere
  raw data: rLLRLrLRlrLrLLRLrLRlrL
  RL: R^2
  AB: ABBABB
```

ThreeBraid クラスにおける 3-braid の表現形式の間の変換は、多くの場合 AB 型表現形式を経由しています。一旦, AB 型表現形式に変換し、AB 型表現形式で簡約化します。変換規則に則って、簡約化された AB 型表現形式から目的の表現形式に変換します。ただし、Syzygy 列への変換に対してのみ、簡約化する前の AB 型表現形式から変換します。AB 型表現形式の簡約化は、簡約則 AA=BBB=id を適応することです。**closure**, **loop** など circular word の場合は、共役類の中で最小語数のものとし、文字 A を含む場合は A を先頭語に、文字 B を含む場合は B を末尾語になるように正規化しています。

Lissajous 3-braid などのように、**loop** が同じ形の３つの部分に分けられるときには、自動的に３分割したものを計算します。さらに３分割したものが更にふたつのブロックに分けられる時は、６分割したものを計算します。ただし、後の方は同じ形の２つとは限りません。鏡映などで対応する場合もあります。**show()** メソッドでそれらを表示させるには、引数 **brStyle** に **'loop3'** や **'loop6'** を指定します。**show(brStyle='all')** などでの **Braid Style: loop /3** や **Braid Style: loop /6** がそれにあたります。

```
        >>> br.set('012012', silent=True)       # (figure-8)
```

```
        >>> br.show(brStyle=['loop', 'loop3'])
```

```
Braid Style: loop on shape sphere
  AB: ABABBABABBABABB
  Sigma: σ1^-1 σ2 σ1^-1 σ2 σ1^-1 σ2
  Syzygy: 012012
  BDPQ (Frieze): qbqbqb
  RL: L R L R L R
  ST: ST3 ST3 ST3
  PSL2: [[5, 8], [8, 13]]
  Dilatation: (18 + sqrt(320)) / 2  (value=17.94427190999916)
Braid Style: loop/3 on shape sphere
  AB: ABABB
  Sigma: σ1^-1 σ2
  Syzygy: 01
  BDPQ (Frieze): qb
  RL: L R
  ST: ST3
  PSL2: [[1, 1], [1, 2]]
  Dilatation: (3 + sqrt(5)) / 2  (value=2.618033988749895)
```

```
        >>> br.set('qbqbqbbbb', silent=True)    # (これも、figure-8)
```

```
        >>> br.set(brType=['raw', 'Frieze', 'AB'])
```

```
Braid Style: braid
```

```
  raw data: qbqbqbbbb
  BDPQ (Frieze): qbqbqb
  AB: ABABBABABBABABB
Braid Style: circular word
  raw data: qbqbqbbbb
  BDPQ (Frieze): qbqbqb
  AB: ABABBABABBABABB
Braid Style: loop on shape sphere
  raw data: qbqbqbbbb
  BDPQ (Frieze): qbqbqb
  AB: ABABBABABBABABB
Braid Style: loop/3 on shape sphere
  raw data: ABABB
  BDPQ (Frieze): qb
  AB: ABABB
Braid Style: loop/6 on shape sphere
  raw data: AB
  BDPQ (Frieze): q
  AB: AB
```

PSL2 型表現形式で与えられた場合、基本変形により S、T （PSL2(Z)=<S, T>）で表し、ST 型表現形式から AB 型表現形式に変換し、他の表現形式に変換します。

```
>>> br.set([[17,4],[4,1]], silent=True)
```

```
>>> br.show(brType=['raw', 'PSL2', 'AB'])
```

```
Braid Style: braid
  raw data: TTTTStttttS
  PSL2: [[17, 4], [4, 1]]
  AB: ABBABBABBABBABABABAB
Braid Style: circular word
  raw data: TTTTStttttS
  PSL2: [[17, 4], [4, 1]]
  AB: ABBABBABBABBABABABAB
Braid Style: loop on shape sphere
  raw data: TTTTStttttS
  PSL2: [[17, 4], [4, 1]]
  AB: ABBABBABBABBABABABAB
```

PSL2(Z) に属さない２次正方行列が与えられた場合は、error メッセージが返されます。

```
>>> br.set([[15,4],[4,1]])
```

```
error: [[15, 4], [4, 1]] is not in PSL2.
```

show() メソッドの紹介の最初で、**ThreeBraid** クラス型オブジェクトの内部インスタンス変数を直接呼び出すことで、必要なデータを文字列として抽出することができます。内部インスタンス変数の名称は、**brStyle** の指定に対応した **braid**, **closure**, **loop**, **loop3**, **loop6** です。それらは辞書型オブジェクトで、3-braid の表現形式に対応した **'AB'**, **'Sigma'**, **BDPQ**, **'Frieze'**, **'Syzygy'**, **'RL'**, **'ST'**, **'PSL2'** と、**'Dilatation'** などのキーワードに対する値として収められています。内部インスタンス変数における 3-braid の表示はやや可視性に劣ります。**ThreeBraid** クラスの **show()** メソッドなどでは、**tb.pprint** 関数を表示に利用しています。これらを使った例は次章 **[LinearBraid クラス]** の中で紹介します。

[コピー] copy() メソッドを使って、**ThreeBraid** クラス型オブジェクトの複製を生成することができます。python において、変数への代入は、新たなオブジェクトを生成して代入するのではなく、同一オブジェクトへの

アクセスポインタの代入なので、一方の変数へ加えられた操作は別の変数にも作用します。計算途中の状態を保持したままで、新たに何らかの操作を加えた結果も参照したい場合など、**copy()** メソッドにより **ThreeBraid** クラス型オブジェクト変数の複製を生成し、新たな変数に割り当てます。この場合、それぞれの変数には異なるオブジェクトが割り当てられ、それらの一方に加えられた変更は他方に影響を与えません。具体的に説明します。

```
>>> br1 = tb.ThreeBraid('ABABB')

>>> br2 = br1
```

最初のコマンドで、初期値 **'ABABB'** を与えて **ThreeBraid** クラス型オブジェクトを生成し変数 **br1** に割り当てています。次のコマンドでは **br1** 変数と同じオブジェクトを別の変数 **br2** に割り当てます。**show()** メソッドでそれぞれの変数に代入された 3-braid をみてみます。

```
>>> br1.show('braid','AB')


Braid Style: braid
  AB: ABABB

>>> br2.show('braid','AB')


Braid Style: braid
  AB: ABABB
```

どちらも同じ 3-braid が代入されています。最初の **br1**に別の 3-braid を代入してみます。

```
>>> br1.set('BABA', silent=True)


>>> br1.show('braid','AB')


Braid Style: braid
  AB: BABA
```

もう一方の変数をみて見ると、

```
>>> br2.show('braid','AB')


Braid Style: braid
  AB: BABA
```

br1 だけでなく **br2** にも新たな値が代入されてしまいました。複製を作る **copy()** メソッドを利用してみます。

```
>>> br3 = br1.copy()


>>> br1.show('braid','AB')


Braid Style: braid
  AB: BABA

>>> br3.show('braid','AB')


Braid Style: braid
  AB: BABA
```

br3 に **br1** と同じ 3-braid が代入されています。

```
>>> br1.set('ABBAB', silent=True)
```

最初の **br1** に新たな値 **'ABBAB'** を与えます。

```
>>> br1.show('braid','AB')
```

```
Braid Style: braid
  AB: ABBAB
```

```
>>> br3.show('braid','AB')
```

```
Braid Style: braid
  AB: BABA
```

br1 には新たな値が代入されましたが、**br3** はもとのままです。ところで、変数 **br1, br2, br3** を呼び出すと、その変数に割り当てられたオブジェクトへのアクセスポインタが返されます。

```
>>> br1
```

```
<ThreeBraid.ThreeBraid object at 0x103e91030>
```

```
>>> br2
```

```
<ThreeBraid.ThreeBraid object at 0x103e91030>
```

```
>>> br3
```

```
<ThreeBraid.ThreeBraid object at 0x103e90fd0>
```

br1 と **br2** が同一オブジェクトを指し、**br3** は **br1, br2** とは異なるオブジェクトを指していることがわかります。

ちなみに、**print** 文や **str** 文の中で呼び出すと、クラス名と 3-braid を定義したときの表現形式が返されます。

```
>>> print(br1)
```

```
ThreeBraid defined by the sequence "ABBAB"
```

[3-braid の連結（加法演算）] いくつかの 3-braid をつなぐ、加法演算もサポートしています。通常の数や式と同じ様に、**'+ '** 記号でつなぐだけです。予め、**Threebraid** クラス型オブジェクト変数を用意し **'+ '** でつないぐこともできますが、変数に割り当てずに、引数を与えて生成した **ThreeBraid** クラス型オブジェクトに対して加法を行うこともできます。

例えば、AB 型表現形式が **'ABBABBABBABBABABABAB'** の 3-braid に **'BBABABABA'** の 3-braid をつないでみます。それぞれを変数に割り当てて、

```
>>> br1 = tb.ThreeBraid('ABBABBABBABBABABABAB')
```

```
>>> br2 = tb.ThreeBraid('BBABABABA')
```

```
>>> br3 = br1 + br2
```

```
>>> br3.show()
```

により、**br1** に **br2** をつないだ 3-braid が **br3** に割り当てられます。ここでの **show()** メソッドの出力は省きます。変数に割り当てる必要がなく、計算結果のみを知りたい場合は、次でじゅうぶんです。

```
        >>> (tb.ThreeBraid('ABBABBABBABBABABABAB') + tb.ThreeBraid('BBABABABA')).show(brType='AB')
```

異なる表現形式で与えられた 3-braid に対しても、連結が計算できます。AB 型表現形式にしてから計算しています。

[3-braid の比較] 3-braid を比較するとき、上端下端のある **braid** として比較するか、上端下端を結んだ **closure** として比較するかで、意味が変わります。**braid** としての比較結果と **closure** としての比較結果の組（tuple）を、3-braid の比較の戻り値としています。比較は、等号（==)、不等号（!=)、大小関係（<, >, <=, >=）からなります。等号、不等号は AB 型表現形式が一致するかどうか、大小関係は AB 型表現形式における包含関係の有無で判定しています。**closure** の場合は巡回語として比較しています。

```
        >>> tb.ThreeBraid('ABABB') == tb.ThreeBraid('qb')
{'braid': True, 'circular': True}
        >>> tb.ThreeBraid('ABABB') == tb.ThreeBraid('BBABA')
{'braid': False, 'circular': True}
        >>> tb.ThreeBraid('ABABB') != tb.ThreeBraid('BBABA')
{'braid': True, 'circular': False}
        >>> tb.ThreeBraid('BABABB') < tb.ThreeBraid('ABABB')
{'braid': False, 'circular': True}
        >>> tb.ThreeBraid('BABABB') > tb.ThreeBraid('ABABB')
{'braid': True, 'circular': False}
```

[LinearBraid クラス]
線形な有限長の syzygy 列をもつ 3-braid で shape sphere 上の loop に対応するものを linear 3-braid といいます。Lissajous 3-braid は linear 3-braid で、linear 3-braid は Lissajous 3-braid で表せるか、周期部分の長さを２倍に伸ばすことで Lissajous 3-braid で表せます。**LinearBraid** クラスは、linear 3-braid, Lissajous 3-braid を扱うための環境で、**ThreeBraid** クラスを拡張したクラスです。**LinearBraid** クラスは Syzygy 列を計算の基本にしています。linear 3-braid を特徴付ける linear paramater か、Lissajous 3-braid を特徴付ける Lissajous type から Syzygy 列を生成し、3-braid を定めます。他の表現形式への変換などに **ThreeBraid** クラスを利用します。**LinearBraid** クラスの利用手順も、**ThreeBraid** クラスと同じ３ステップ **[インスタンスの生成]**、**[パラメーターの入力]**、**[結果出力]** です。

[インスタンスの生成]　　**LinearBraid** クラス型オブジェクトを生成し、変数 **lb** に割り当てます。

```
        >>> lb = tb.LinearBraid()
```

[パラメーターの入力]　　Linear 3-braid は、linear paramater で特徴付けられます。linear paramater は傾き、切片、長さの３つ組です。傾きは有理数、切片は実数、長さは非負偶数で、傾きと長さの積は

３の倍数です。Lissajous 3-braid は、Lissajous type で特徴付けられます。Lissajous type は２つの振動数と、フェイズの３つ組です。振動数は３と素な整数で、フェイズは実数です。**LinearBraid** クラス型オブジェクトへのパラメーターの入力は、**set()** メソッドのキーワード引数（**linear**, **lissajous**, **lissajousnew**）への割り当てで行います。例えば、linear paramater として傾き１、切片1/2、語長６を指定する場合、

```
>>> lb.set(linear=[1, '1/2', 6])
```

とします。Lissajous type **[4, -5, 0]**（振動数 4, -5, フェイズパラメータ 0）を指定する場合は、

```
>>> lb.set(lissajous=[4, -5, 0])
```

とします。

（注意） **LinearBraid** クラスでは、Lissajous 曲線として

$$L(t) = \sin(2\pi\, m(t + \delta)) + i \sin(2\pi\, n\, t) \qquad 0 \leqq t \leqq 1$$

で与えられる閉曲線をとり、三つ組 **[m, n, δ]** を Lissajous Type としています。後で具体的に定義式を与えますが、[KNO2] とは Lissajous 曲線のパラメーターが少し変違っています。上記閉曲線では実部に置かれたフェイズパラメータですが、[KNO2] では虚部に移しています。変更後の三つ組で計算する場合のために、**set()** メソッドのキーワード引数 **lissajousnew** を用意しています。

```
>>> lb.set(lissajousnew=[4, -5, 0])
```

のように指定します。上記 Lissajous 曲線の型 **[m, n, δ]** に対応する、（変更後の）Lissajous 曲線の型は **[-n, m, 2π mδ]** で、上記 Lissajous 曲線を π/2 回転したものになっています。

Linear 3-braid のlinear paramater において、３つ目の語長が略された場合は、傾きの分母の６倍で補われます。さらに２つ目も略されて、傾きのみが長さが1のリストか有理数で与えられた場合、２つ目の切片には傾きの分母の２倍を分母とする単位分数が補われ、３つ目の語長には傾きの分母の６倍が指定されます。この意味で次はすべて同じパラメーター指定です。

```
>>> lb.set(linear=[1, '1/2', 6])
>>> lb.set(linear=[1, '1/2'])
>>> lb.set(linear=[1])
>>> lb.set(linear=1)
```

切片が指定されなかったときに傾きの分母の２倍を分母とする単位分数を補うのは、collision-free にするためです。Linear paramater における傾き a, 切片 b の Linear braid が collision-free であるための必要十分条件は、直線 y=ax+ b が格子点を通らないことです。

ところで、すでに使っている表記ですが、引数に有理数を指定するときに、**'1/2'** のように文字列を使います。プログラム内部で有理数を扱うときに、**fractions** モジュールの **Frantion** クラスを利用しています。有理数 **q/p** は、**Fraction(q, p), Fraction('q/p')** で表されます。**ThreeBraidClass** モジュールでの引数などの入力に分数を使う場合、**fractions** モジュールを組み込んで **Fraction** クラスを使うこともできますが、文字列 **'q/p'** で受け渡すこともできます。**ThreeBraidClass** モジュールでは、後者の文字列表記を推奨します。

LinearBraid クラスは **ThreeBraid** クラスを拡張したものなので、**LinearBraid** クラスで **ThreeBraid** クラスを代用することができます。**set()** メソッドで、キーワード引数（**linear, lissajous, lissajousnew**）に値を割り当てず、引数を与えて呼び出したとき、**ThreeBraid** クラスを代用していると解釈し、**ThreeBraid** クラスにおける **set()** メソッドが呼び出され、新たに **set()** メソッドで linear 3-braid が代入されるまで **ThreeBraid** クラスとして機能します。

ThreeBraid クラスと同様に、**LinearBraid** クラスもインスタンス生成の際に引数を指定することで、インスタンス生成と入力を一度に行うことができます。

[出力] **LinearBraid** クラス型オブジェクト変数の内容を表示するために、**show()** メソッドを使います。**show()** メソッドは、linear 3-braid としてのパラメータ（linear paramater と Lissajous type）と、パラメータから生成された AB 型表現形式と Syzygy 列、それらの linear cancellation、簡約形を出力します。**show()** メソッドは、**ThreeBraid** クラスの **shoq()** メソッドと同じキーワード引数 （ **brSty**

le, brType）をもち、それらの少なくとも一つを指定することで、LinearBraid クラスに埋め込まれた ThreeBraid クラスの show() メソッドが呼び出されます。キーワード引数（ brStyle, brType）の指定方法も、表示される内容も、ThreeBraid クラスの show() メソッドと同じです。

```
        >>> lb.set(linear=[1, '1/2', 6], silent=True)
        >>> lb.show()

Linear parameter: [1, 1/2, 6]
Lissajous Type: [-1, -4, 0]
  Collision-free?: True
Raw Data
  AB3: ABBAB
  Syzygy3: 01
Linear cancellation       slope: 1
  AB3: ABBAB
  Syzygy3: 01
Reduced sequences   (1/3)
  AB: ABBAB
  Syzygy: 01
  BDPQ (Frieze): dp
  PSL2: [[2, 1], [1, 1]]
  Dilatation: (3 + sqrt(5)) / 2  (value=2.618033988749895)
        >>> lb.show(brStyle='loop3', brType=['AB', 'Syzygy', 'RL', 'PSL2', 'Dilatation'])

Braid Style: loop/3 on shape sphere
  AB: ABBAB
  Syzygy: 01
  RL: R L
  PSL2: [[2, 1], [1, 1]]
  Dilatation: (3 + sqrt(5)) / 2  (value=2.618033988749895)
```

linear paramater として [1, '1/2', 6] ではなく、[1, '1/2', 12] とすると、syzygy 列の語長が２倍になるので、3-braid も２倍になり、PSL2 表示と dilatation は２乗になります。

```
        >>> lb.set(linear=[1, '1/2', 12], silent=True)
        >>> lb.show(brStyle='loop3', brType=['AB', 'Syzygy', 'RL', 'PSL2', 'Dilatation'])

Braid Style: loop/3 on shape sphere
  AB: ABBABABBAB
  Syzygy: 0120
  RL: R L R L
  PSL2: [[5, 3], [3, 2]]
  Dilatation: (7 + sqrt(45)) / 2  (value=6.854101966249685)
```

Linear 3-braid, Lissajous 3-braid は必ず同じ形の３つの部分に分けられるので、show() メソッドでは３分割したものを出力しています。ThreeBraid クラスにおける loop3 に対応するものです。show() メソッドでの出力は、'Raw data', 'linear cancellation', 'reduced sequences' の３項目に分かれています。'Raw data' には set() メソッドで入力された linear paramater や Lissajous type から生成された素データが、'linear cancellation' には linear cancellation で得られた傾きをもとに生成されたものが、'reduced sequences' には素データを簡約化したものが表示されます。

```
        >>> lb.set(linear=['1/5'], silent=True)
        >>> lb.show()

Linear parameter: [1/5, 1/10, 30]
Lissajous Type: [-1, -16, -13/32]
  Collision-free?: True
Raw Data
  AB3: AAAAABAAAAABB
  Syzygy3: 0000011111
Linear cancellation       slope: 1
  AB3: ABABB
  Syzygy3: 01
Reduced sequences   (1/3)
```

```
  AB: ABABB
  Syzygy: 01
  BDPQ (Frieze): qb
  PSL2: [[1, 1], [1, 2]]
  Dilatation: (3 + sqrt(5)) / 2  (value=2.618033988749895)
```

キーワード引数（**linear**, **lissajous**, **lissajousnew**）を指定せずに **set()** メソッドを呼び出したとき、**LinearBraid** クラスは **ThreeBraid** クラスを代用していると解釈し、**ThreeBraid** クラスとして扱われます。このとき、出力メソッド **show()** は **ThreeBraid** クラスの **show()** メソッドが使われます。

```
        >>> lb.set('ABBABABBABABBAB', silent=True)
        >>> lb.show()

Braid Style: braid
  AB: ABBABABBABABBAB
  Sigma: σ2 σ1^-1 σ2 σ1^-1 σ2 σ1^-1
  Syzygy: 0210210
  BDPQ (Frieze): dpdpdp
  RL: R L R L R L
  ST: T2 ST3 ST3 ST
  PSL2: [[13, 8], [8, 5]]
  Dilatation: (18 + sqrt(320)) / 2  (value=17.94427190999916)
Braid Style: circular word
  AB: ABBABABBABABBAB
  Sigma: σ2 σ1^-1 σ2 σ1^-1 σ2 σ1^-1
  Syzygy: 021021
  BDPQ (Frieze): dpdpdp
  RL: R L R L R L
  ST: ST3 ST3 ST3
  PSL2: [[13, 8], [8, 5]]
  Dilatation: (18 + sqrt(320)) / 2  (value=17.94427190999916)
Braid Style: loop on shape sphere
  AB: ABBABABBABABBAB
  Sigma: σ2 σ1^-1 σ2 σ1^-1 σ2 σ1^-1
  Syzygy: 021021
  BDPQ (Frieze): dpdpdp
  RL: R L R L R L
  ST: ST3 ST3 ST3
  PSL2: [[13, 8], [8, 5]]
  Dilatation: (18 + sqrt(320)) / 2  (value=17.94427190999916)
Braid Style: loop/3 on shape sphere
  AB: ABBAB
  Sigma: σ2 σ1^-1
  Syzygy: 02
  BDPQ (Frieze): dp
  RL: R L
  ST: ST3
  PSL2: [[2, 1], [1, 1]]
  Dilatation: (3 + sqrt(5)) / 2  (value=2.618033988749895)
Braid Style: loop/6 on shape sphere
  AB: ABB
  Sigma: 2
  Syzygy: 0
  BDPQ (Frieze): d
  RL: R
  ST: ''
  PSL2: [[-1, -1], [0, -1]]
```

[Lissajous 3-braid as linear 3-braid]　Lissajous 3-braid は linear 3-braid です。Lissajous 3-braid を **LinearBraid** クラスで計算する場合も、**set()** メソッドで Lissajous type を入力し、**show()** メソッドで出力するだけです。Lissajous type は **[整数, 整数, 実数]** の３つ組で与えます。

```
        >>> lb.set(lissajous=[-1,4,Fraction('-5/16')], silent=True)
        >>> lb.show()
```

```
Linear parameter: [-2, -3, 6]
Lissajous Type: [-1, 4, -5/16]
  Collision-free?: False
Raw Data
  AB3: ABBAB
  Syzygy3: 01
Linear cancellation       slope: 1
  AB3: ABBAB
  Syzygy3: 01
Reduced sequences   (1/3)
  AB: ABBAB
  Syzygy: 01
  BDPQ (Frieze): dp
  PSL2: [[2, 1], [1, 1]]
  Dilatation: (3 + sqrt(5)) / 2  (value=2.618033988749895)
```

計算例を少し。

```
        >>> lb.set(lissajous=[4,-11,0], silent=True)
        >>> lb.show()

Linear parameter: [11/5, -5/2, 30]
Lissajous Type: [4, -11, 0]
  Collision-free?: True
Raw Data
  AB3: ABABBAABBABABBABAABABB
  Syzygy3: 0211021002
Linear cancellation       slope: -4/3
  AB3: ABBABBABABABABB
  Syzygy3: 010202
Reduced sequences   (1/3)
  AB: ABABABABBABBABB
  Syzygy: 020212
  BDPQ (Frieze): qpqbdb
  PSL2: [[1, 3], [3, 10]]
  Dilatation: (11 + sqrt(117)) / 2  (value=10.908326913195985)

        >>> lb.set(lissajous=[4,19,0], silent=True)
        >>> lb.show()

Linear parameter: [11/5, -5/2, 30]
Lissajous Type: [4, 19, 0]
  Collision-free?: True
Raw Data
  AB3: ABABBAABBABABBABAABABB
  Syzygy3: 0211021002
Linear cancellation       slope: -4/3
  AB3: ABBABBABABABABB
  Syzygy3: 010202
Reduced sequences   (1/3)
  AB: ABABABABBABBABB
  Syzygy: 020212
  BDPQ (Frieze): qpqbdb
  PSL2: [[1, 3], [3, 10]]
  Dilatation: (11 + sqrt(117)) / 2  (value=10.908326913195985)

        >>> lb.set(lissajous=[4,-23,0], silent=True)
        >>> lb.show()

Linear parameter: [23/9, -5/2, 54]
Lissajous Type: [4, -23, 0]
  Collision-free?: True
Raw Data
```

```
  AB3: AABBAABBAABBAABBAAABAABAABAABA
  Syzygy3: 002211002221100221
Linear cancellation       slope: -1
  AB3: ABBAB
  Syzygy3: 02
Reduced sequences   (1/3)
  AB: ABABB
  Syzygy: 21
  BDPQ (Frieze): qb
  PSL2: [[1, 1], [1, 2]]
  Dilatation: (3 + sqrt(5)) / 2  (value=2.618033988749895)
```

Lissajous 3-braid において、振動数を固定し、フェイズパラメータを変化させて、Syzygy 列の移り変わりをみてみましょう。

```
        >>> for d in range(10):
                delta = (' '+ str(d) + '/10')[-5:]
                lb.set(lissajous=[4,-5,delta], silent=True)
                print(' {} [{}] {}'.format(delta, lb.startArea, lb.RawData['Syzyg
y']))

         0/10 [0-] 020212101020212101
         1/10 [1+] 102021210102021210
         2/10 [1+] 101020212101020212
         3/10 [1+] 121010202121010202
         4/10 [0+] 021210102021210102
         5/10 [0+] 020212101020212101
         6/10 [1-] 102021210102021210
         7/10 [1-] 101020212101020212
         8/10 [1-] 121010202121010202
         9/10 [0-] 021210102021210102
```

フェイズパラメータをもっと細かく変化させて、Syzygy 列が変わったところのみを表示させてみましょう。

```
        >>> wk = ''
        >>> for d in range(-5, 410):
                delta = ('  '+ str(d) + '/400')[-7:]
                lb.set(lissajous=[4,-5,delta], silent=True)
                if wk == '': wk = lb.RawData['Syzygy']
                if wk != lb.RawData['Syzygy']:
                    print('     {} [{}] {}:{}'
                          .format(
                              delta, lb.startArea,
                              lb.RawData['Syzygy'],
                              tb.compCW(lb.RawData['Syzygy'], wk, simple=False)[-1]
                          )
                         )
                    wk = lb.RawData['Syzygy']

          1/400 [2+] 202121010202121010:1
         21/400 [1+] 102021210102021210:7
         61/400 [1+] 101020212101020212:7
        101/400 [1+] 121010202121010202:7
        141/400 [0+] 021210102021210102:7
        181/400 [0+] 020212101020212101:7
        201/400 [2-] 202121010202121010:1
        221/400 [1-] 102021210102021210:7
        261/400 [1-] 101020212101020212:7
        301/400 [1-] 121010202121010202:7
        341/400 [0-] 021210102021210102:7
        381/400 [0-] 020212101020212101:7
        401/400 [2+] 202121010202121010:1
```

同じことを AB 型表現形式で見てみます。

```
>>> wk = ''
>>> for d in range(-5, 410):
        delta = ('  '+ str(d) + '/400')[-7:]
        lb.set(lissajous=[4,-5,delta], silent=True)
        if wk == '': wk = lb.RawData['AB']
        if wk != lb.RawData['AB']:
            print(
                '       {} [{}] {}:{}'
                .format(
                    delta,
                    lb.startArea, lb.RawData['AB'],
                    tb.compCW(lb.RawData['AB'], wk, simple=False)[-1]
                )
            )
            wk = lb.RawData['AB']

  1/400 [2+] ABABABBABBABBABABABABBABBABBABABABABBABBABBAB:2
 21/400 [1+] ABBABABABABBABBABBABABABABBABBABBABABABABBABB:10
 61/400 [1+] ABBABBABBABABABABBABBABBABABABABBABBABBABABAB:9
101/400 [1+] ABABABBABBABBABABABABBABBABBABABABABBABBABBAB:11
141/400 [0+] ABBABABABABBABBABBABABABABBABBABBABABABABBABB:10
181/400 [0+] ABBABBABBABABABABBABBABBABABABABBABBABBABABAB:9
201/400 [2-] ABBABBABABABABBABBABBABABABABBABBABBABABABABB:3
221/400 [1-] ABABBABBABBABABABABBABBABBABABABABBABBABBABAB:10
261/400 [1-] ABABABABBABBABBABABABABBABBABBABABABABBABBABB:11
301/400 [1-] ABBABBABABABABBABBABBABABABABBABBABBABABABABB:9
341/400 [0-] ABABBABBABBABABABABBABBABBABABABABBABBABBABAB:10
381/400 [0-] ABABABABBABBABBABABABABBABBABBABABABABBABBABB:11
401/400 [2+] ABABABBABBABBABABABABBABBABBABABABABBABBABBAB:2
```

LinearBraid クラスは Syzygy 列を基盤にしています。syzygy 列が shape sphere 上の道が赤道を通過する様子（３点が一直線上に並ぶ朔の状態）の移り変わりを表すものなので、赤道に対して対称な道も同じ syzygy 列をもちます。赤道に対して対称な道を区別するために、道の始点が北半球にあるか南半球にあるか、もし赤道上であれは北半球に向かっているか南半球に向かっているかを指定します。linear 3-braid は、明示的に始点を指定していない限り、南半球とし、syzygy 列の最初の文字の示す赤道上の区画から北半球に向かうことにしています。Lissajous 3-braid の場合は、パラメータ t=0 での３点の配置に対する shape sphere 上の点を始点としています。**ThreeBraid** クラスにおいて、対応する shape sphere 上の道は、syzygy 列の最初の文字の示す赤道の区画を始点とし、sygyzy 列の示す通りに赤道を通過し、syzygy 列の最後の文字の示す赤道の区画を終点とするものを想定しています。始点を指定する **startArea** の最初の文字と競合することがありますが、syzygy 列の記述を優先しています。**startArea** の符号により、道の向かう方向を定めることで、shape sphere 上の道がただ一つに定まります。その後、shape sphere の６つの領域の通過状況から AB 型表現形式に変換し、AB 型表現形式から他の表現形式に変換しています。AB 型変換形式を syzygy 列に変換する場合は、**startArea** で指定した syzygy の状態と向きを始点とし AB 型表現形式に従って shape sphere 上の６つの領域の通過順と、赤道の通過の様子、つまり syzygy 列が定まることになります。この関係では、AB 型表現形式で文字 **'A'** に対応するところで赤道を横切り、syzygy の状態になりますから、syzygy 列の長さは、AB 型表現形式での文字 **'A'** の個数に始点の分の１を加えた数になります。Linear 3-braid や Lissajous 3-braid、あるいは **ThreeBraid** クラスにおける **loop** の場合は、shape sphere 上の閉路に対応しているため、始点と終点が一致します。従って、この場合の syzygy 列の長さは、AB 型表現形式での文字 **'A'** の個数に等しくなります。

[始点の移動]　　**LinearBraid** クラスで扱う linear 3-braid は、線形な Syzygy 列をもつ shape sphere 上の閉路です。閉路の始点を閉路上の任意の点に移動させると、円環状に並んだ文字列としての Syzygy 列の始点が変わります。Syzygy 列の始点の移動には、**shift()** メソッドを使います。**shift()** メソッドは始点を移動させる文字数を表す整数値引数をひとつ持ちます。引数が省略された場合は、１を指定したとみなします。正の整数の場合は前方（文字列表記における右方向）に、負の整数の場合は後方（文字列表記における左方向）に始点を移動させます。

```
>>> lb.set(linear='-4/3', silent=True)
>>> lb.show(brStyle='loop', brType=['Syzygy', 'AB'])
```

```
Braid Style: loop on shape sphere
  Syzygy: 012120201012120201
  AB: ABABBABBABBABABABABBABBABBABABABABBABBABBABAB
        >>> lb.shift(3)
        >>> lb.show(brStyle='loop', brType=['Syzygy', 'AB'])

Braid Style: loop on shape sphere
  Syzygy: 120201012120201012
  AB: ABABBABBABBABABABABBABBABBABABABABBABBABBABAB
```

[連結（加法演算）] 加法演算により、**LinearBraid** クラスをつなぐことができます。**LinearBraid** クラスの加法は、始点で切り分けた Syzygy 列を連結し、連結した Syzygy 列によって定まる **ThreeBraid** クラスを戻り値としています。始点をどこに取るかによって、結合されたものが異なります。結合させる位置を指定するために、**shift()** メソッドで予め始点を変更しておく必要があます。また、**ThreeBraid** クラスとの結合（加法）も定義されています。**LinearBraid** クラスの方は始点で切り分けておいて、**ThreeBraid** クラス同士の結合（加法）として AB 型表現形式をつないで与えられる **ThreeBraid** クラスを戻り値としています。

[比較] **LinearBraid** クラスの間の比較演算は、shape sphere 上の閉路として、巡回語の比較で定義されます。**ThreeBraid** クラスは、上端下端のある **braid** と、上端下端をつないだ **closure** のふたつの立場での比較でした。**LinearBraid** クラスと **ThreeBraid** クラスの比較演算も定義されます。この場合、**LinearBraid** クラスを始点で切り分けて **ThreeBraid** クラスと読み替えて比較します。

[LissajousBraid class]
LissajousBraid クラスは、Lissajous 3-braid に関する [KNO1]/[KNO2] で用いられている様々な量を計算するためのクラスです。このクラスにおける Lissajous 曲線は、**LinearBraid** クラスと若干異なる次の形のものとします。

$$L(t) = \sin(2\pi\, m\, t) + i \sin(2\pi\, n\, t + \varphi) \qquad 0 \leqq t \leqq 1$$

Lissajous type **[m, n, φ]**（整数 m, n は３を法として１と合同）は、**LinearBraid** クラスの時と同じ三つ組ですが、フェイズパラメータ φ （**LinearBraid** クラスのときは δ）の位置が異なっています。**LinearBraid** クラスでの、Lissajous type **[n, −m, φ/2πm]** のLissajous 曲線を原点中心に −π/2 回転させると、上の Lissajous 曲線に一致します。**LinearBraid** クラスは syzygy 列を基盤としています。**LissajousBraid** クラスでは BDPQ 語（Frieze pattern）を基盤にしています。[KNO1]/[KNO2] では、main formula が AB 型表現形式で記述されていますが、AB 型表現形式を経由せずに BDPQ 語で記述する手順が与えられ、BDPQ 語を使ってさまざまな話題との関係が描かれています。[KNO1] では、Lissajous type が **[m, n, 0]**（m≡n≡1 mod 3, m−n≡1 mod 2）のみを扱っています。[KNO2] では、3-braid を形成しうる全ての Lissajous type **[m, n, δ]**（m, n が共に３で割り切れず、collision-free でない）を扱うことができます。
LissajousBraid クラスの使い方も、**ThreeBraid** クラス、**LinearBraid** クラスと同じです。**LissajousBraid** クラス型オブジェクト変数を用意し、**set()** メソッドで Lissajous type を指定し、**show()** メソッドで計算結果などを表示します。**set()** メソッドの引数は Lissajous type **[m, n, φ]** です。フェイズパラメータ φ を入力する際に、円周率 π を使いたい場合があります。**math** モジュールを **import** していれば、**math.pi** で円周率を入力することができます。**math** モジュールを **import** していなくても、**tb.pi** により、**ThreeBraidClass** モジュールに組み込まれた円周率を利用できます。

```
        >>> lis = tb.LissajousBraid()
        >>> lis.set([1,4,1])

        >>> lis.show()

Lissajous Type: (1, 4, 1, -1)
    (primitive) (1, -2, (-3.141592653589793, '-1/1 π'))
Lissajous Curve: sin(2πt) + i sin(8πt+1) (t : a real number)

 Slope 0/1  /  Level: 1
- - - - - - - - - - - - - - - - - - - -
  ε': 1001
  ε: +--+
```

```
  W: BABBAABBAB
  V: BABBA
  Wbdpq: pd
  Vbdpq: pd
  PSL2: [[2, -1], [-1, 1]]
  PSL2/V: [[1, 1], [-1, 0]]
  Dilatation: (3 + sqrt(5)) / 2  (value=2.618033988749895)
  Dilatation/V: (1 + sqrt(-3)) / 2  (value=(0.5+0.8660254037844386j))
- - - - - - - - - - - - - - - - - - - - - -
  phi0: (-6.283185307179586, '-2/1 π')
  st_ε': 10
  st_ε: -+
  st_W: BBABA
  st_Wbdpq: pd
  AB: BABBAABBAB
  R: BBABA
  Rbdpq: pd
  st_RL: L^(-1) R^(-1)
- - - - - - - - - - - - - - - - - - - - - -
  Christoffel Word: 0
  Palindromic Conjugate: 0
  δ': 1
  δ'->ε': 01
  r (4.11): (1,)
  V (4.11): d
  R (4.11): dd
```

LissajousBraid クラスは、**LinearBraid** クラス型オブジェクトの内部インスタンス変数 **'lb'** をもちます。**lis.set()** メソッドで **LissajousBraid** クラス型オブジェクト変数に Lissajous Type を定義した時に、**LinearBraid** クラス型オブジェクト変数 **lis.lb** の **setAB()** メソッド（**lis.lb.setAB()**）が呼び出され、Linear 3-braid としての Lissajous 3-braid が **lis.lb** に格納されます。**set()** メソッドで入力された **LinearBraid** クラス型オブジェクトは、Syzygy 列をもとに生成されましたが、**LinearBraid** クラス型オブジェクトの **setAB()** メソッドは、**LissajousBraid** クラス型オブジェクトで計算された AB 型表現形式を使うための特別なメソッドで、**LinearBraid** クラス型オブジェクトへの入力は Syzygy 列をもとにした **set()** メソッドを使ってください。**lis.lb.show()** により、**LinearBraid** クラス型オブジェクト変数 **lis.lb** の **show()** メソッドを呼び出すことができます。

```
        >>> lis.lb.show()

Linear parameter: [2, 2.5795774715459476, 6]
Lissajous Type: [4, -1, 0.039788735772973836]
  Collision-free?: True
Raw Data
  AB3: BABBA
  Syzygy3: 21
Linear cancellation       slope: -1
  AB3: ABBAB
  Syzygy3: 02
Reduced sequences   (1/3)
  AB: ABBAB
  Syzygy: 21
  BDPQ (Frieze): dp
  PSL2: [[2, 1], [1, 1]]
  Dilatation: (3 + sqrt(5)) / 2  (value=2.618033988749895)
```

上で **LinearBraid** クラスで扱ったのと同じ例を **LissajousBraid** クラスで計算してみます。

```
        >>> lis.set([4,-11,0], silent=True)
        >>> lis.show()

Lissajous Type: (4, -11, 0, 5)
    (primitive) (4, -5, (-3.141592653589793, '-1/1 π'))
Lissajous Curve: sin(8πt) + i sin(-22πt+0) (t : a real number)
```

```
 Slope 0/1  /  Level: 2
- - - - - - - - - - - - - - - - - - - - -
  ε': 01101001
  ε: −++−+−−+
  W: ABBABBABBABABBAABBAB
  V: ABBABBABBA
  Wbdpq: dbdpqp
  Vbdpq: dbd
  PSL2: [[10, 3], [3, 1]]
  PSL2/V: [[3, −1], [1, 0]]
  Dilatation: (11 + sqrt(117)) / 2  (value=10.908326913195985)
  Dilatation/V: (3 + sqrt(5)) / 2  (value=2.618033988749895)
- - - - - - - - - - - - - - - - - - - - -
  phi0: (−5.497787143782138, '−7/4 π')
  st_ε': 01001011
  st_ε: −+−−+−++
  st_W: ABBABABBAABBABABBABB
  st_Wbdpq: dpqpdb
  AB: ABBABBABBABABAB
  R: ABBABABBAABBABABBABB
  Rbdpq: dpqpdb
  st_RL: R L^3 R^2
- - - - - - - - - - - - - - - - - - - - -
  Christoffel Word: 0
  Palindromic Conjugate: 0
  δ': 1011
  δ'−>ε': 01101001
  r (4.11): (2,)
  V (4.11): dbd
  R (4.11): dbddbd
```

LinearBraid クラスと、**LissajousBraid** クラス環境下での **LinearBraid** クラスを、AB 型表現形式と Syzygy 列で比較してみます。

```
        >>> lb.set(lissajous=[4,−11,0], silent=True)
        >>> lb.show(brStyle=['RawData', 'LinearCancellation', 'loop3'],
                    brType=['AB', 'Syzygy']

Braid Style: Raw Data
  AB: ABABBAABBABABBABAABABBABABBAABBABABBABAABABBABABBAABBABABBABAABABB
  Syzygy: 021102100210221021102100210221
Braid Style: Linear Cancellation
  AB: ABBABBABABABABBABBABBABABABABBABBABBABABABABB
  Syzygy: 010202121010202121
Braid Style: loop/3 on shape sphere
  AB: ABABABABBABBABB
  Syzygy: 020212

        >>> lis.set([4,−11,0], silent=True)
        >>> lis.lb.show(brStyle=['RawData', 'LinearCancellation', 'loop3'],
                        brType=['AB', 'Syzygy'])


Braid Style: Raw Data
  AB: ABBABBABBABABBAABBABABBABBABBABABBAABBABABBABBABBABABBAABBAB
  Syzygy: 020210021010211021210221O
Braid Style: Linear Cancellation
  AB: ABABABBABBABBABABABABBABBABBABABABABBABBABBAB
  Syzygy: 010202121010202121
Braid Style: loop/3 on shape sphere
  AB: ABBABBABBABABAB
  Syzygy: 020212
```

LinearBraid クラスでは、linear paramater や Lissajous type から Syzygy 列が計算され、型変換によりその Syzygy 列から AB 型表現形式が生成されます。**LissajousBraid** クラス環境下での **LinearBraid** クラスでは、**LissajousBraid** クラスで計算された AB 型表現形式から Syzygy 列が生成されます。その違いは **Raw Data** に現れています。**LinearBraid** クラスでは、Syzygy 列は簡約化されておらず、AB 型表現形式では Syzygy 列の文字の重複に対応して文字 **'A'** の重複が見られます。**LissajousBraid** クラス環境下での **LinearBraid** クラスでは、AB 型表現に文字 **'B'** の重複が見られますが、計算方法の違いから文字 **'A'** の重複は見られません。また、Syzygy 列は AB 型表現形式からの型変換過程でほぼ簡約化されています。いずれにせよ、簡約化後は同じです。

```
        >>> lis.set([4,19,0], silent=True)
        >>> lis.show()

Lissajous Type: (4, 19, 0, -5)
    (primitive) (4, -5, (-3.141592653589793, '-1/1 π'))
Lissajous Curve: sin(8πt) + i sin(38πt+0) (t : a real number)

 Slope 0/1  /  Level: 2
- - - - - - - - - - - - - - - - - - - - - -
  ε': 10010110
  ε: +--+-++-
  W: BABBAABBABABBABBABBA
  V: BABBAABBAB
  Wbdpq: pqpdbd
  Vbdpq: pqp
  PSL2: [[10, -3], [-3, 1]]
  PSL2/V: [[3, 1], [-1, 0]]
  Dilatation: (11 + sqrt(117)) / 2  (value=10.908326913195985)
  Dilatation/V: (3 + sqrt(5)) / 2  (value=2.618033988749895)
- - - - - - - - - - - - - - - - - - - - - -
  phi0: (-6.283185307179586, '-2/1 π')
  st_ε': 10010110
  st_ε: -++-+--+
  st_W: BBABAABABBABABBBBABA
  st_Wbdpq: pqpdbd
  AB: BABABABBABBABBA
  R: BBABAABABBABABBBBABA
  Rbdpq: pqpdbd
  st_RL: L^(-3) R^(-3)
- - - - - - - - - - - - - - - - - - - - - -
  Christoffel Word: 0
  Palindromic Conjugate: 0
  δ': 1011
  δ'->ε': 01101001
  r (4.11): (2,)
  V (4.11): dbd
  R (4.11): dbddbd

        >>> lis.set([4,-23,0], silent=True)
        >>> lis.show()

Lissajous Type: (4, -23, 0, 9)
    (primitive) (1, -2, (-3.141592653589793, '-1/1 π'))
Lissajous Curve: sin(8πt) + i sin(-46πt+0) (t : a real number)

 Slope 0/1  /  Level: 1
- - - - - - - - - - - - - - - - - - - - - -
  ε': 11110000
  ε: ++++----
  W: BBBBABBAABBAABBAABBA
  V: BBBB
  Wbdpq: pd
  Vbdpq: p
```

```
  PSL2: [[2, -1], [-1, 1]]
  PSL2/V: [[1, 1], [-1, 0]]
  Dilatation: (3 + sqrt(5)) / 2  (value=2.618033988749895)
  Dilatation/V: (1 + sqrt(-3)) / 2  (value=(0.5+0.8660254037844386j))
- - - - - - - - - - - - - - - - - - - - -
  phi0: (-10.210176124166829, '-13/4 π')
  st_ε': 00011110
  st_ε: ---++++-
  st_W: ABBAABBAABBABBBBABBA
  st_Wbdpq: dp
  AB: BABBA
  R: ABBAABBAABBABBBBABBA
  Rbdpq: dp
  st_RL: R^(-1) L^(-1)
- - - - - - - - - - - - - - - - - - - - -
  Christoffel Word: 0
  Palindromic Conjugate: 0
  δ': 1
  δ'->ε': 01
  r (4.11): (1,)
  V (4.11): d
  R (4.11): dd

        >>> lis.set([-5,7,11/10*math.pi], silent=True)
        >>> lis.show()

Lissajous Type: (-5, 7, 3.455751918948773, -4)
    (primitive) (-5, 7, (3.455751918948773, '11/10 π'))
Lissajous Curve: sin(-10πt) + i sin(14πt+3.455751918948773) (t : a real number)

 Slope 1/1  /  Level: 1
- - - - - - - - - - - - - - - - - - - - -
  ε': 0100101001
  ε: +-++-+-++-
  W: ABABBABAABABBABABBABAABABB
  V: ABABBABAABABB
  Wbdpq: qbdbqbdb
  Vbdpq: qbdb
  PSL2: [[4, 15], [5, 19]]
  PSL2/V: [[1, 3], [1, 4]]
  Dilatation: (23 + sqrt(525)) / 2  (value=22.9564392373896)
  Dilatation/V: (5 + sqrt(21)) / 2  (value=4.7912878474779195)
- - - - - - - - - - - - - - - - - - - - -
  phi0: (3.455751918948773, '11/10 π')
  st_ε': 0100101001
  st_ε: -+--+-+--+
  st_W: ABBABABBAABBABABBABABBAABBAB
  st_Wbdpq: qbdbqbdb
  AB: ABABBABBABBABABBABBABB
  R: ABBABABBAABBABABBABABBAABBAB
  Rbdpq: qbdbqbdb
  st_RL: R L^3 R L^3
- - - - - - - - - - - - - - - - - - - - -
  Christoffel Word: 01
  Palindromic Conjugate: 10
  δ': 11011
  δ'->ε': 0100101001
  r (4.11): (2, 1)
  V (4.11): bdbq
  R (4.11): bdbqbqpq

        >>> lis.set([-11,13,23/22*math.pi], silent=True)
        >>> lis.show()
```

```
Lissajous Type: (-11, 13, 3.2843923196620564, -8)
    (primitive) (-11, 13, (3.2843923196620564, '23/22 π'))
Lissajous Curve: sin(-22πt) + i sin(26πt+3.2843923196620564) (t : a real number)

 Slope 1/1  /  Level: 2
- - - - - - - - - - - - - - - - - - - - -
  ε': 0110100100101101001001
  ε: +--+-++-++-+--+-++-++-
  W: ABABBBBABABBABAABABBABAABABBABABBBBABABBABAABABBABAABABB
  V: ABABBBBABABBABAABABBABAABABB
  Wbdpq: qpqbdbdbqpqbdbdb
  Vbdpq: qpqbdbdb
  PSL2: [[16, 85], [51, 271]]
  PSL2/V: [[1, 5], [3, 16]]
  Dilatation: (287 + sqrt(82365)) / 2  (value=286.99651563714013)
  Dilatation/V: (17 + sqrt(285)) / 2  (value=16.940971508067065)
- - - - - - - - - - - - - - - - - - - - -
  phi0: (3.2843923196620564, '23/22 π')
  st_ε': 011010010010110100 1001
  st_ε: -++-+--+--+-++-+--+--+
  st_W: ABBABBABBABABBAABBABABBAABBABABBABBABBABABBAABBABABBAABBAB
  st_Wbdpq: qpqbdbdbqpqbdbdb
  AB: ABABABABBABBABBABBABABABABBABBABBABBABB
  R: ABBABBABBABABBAABBABABBAABBABABBABBABBABABBAABBABABBAABBAB
  Rbdpq: qpqbdbdbqpqbdbdb
  st_RL: R^3 L^5 R^3 L^5
- - - - - - - - - - - - - - - - - - - - -
  Christoffel Word: 01
  Palindromic Conjugate: 10
  δ': 10111011011
  δ'->ε': 011010010010110100 1001
  r (4.11): (3, 2)
  V (4.11): bdbdbqpq
  R (4.11): bdbdbqpqbdbqpqpq

        >>> lis.set([-17,19,35/34*math.pi], silent=True)
        >>> lis.show()

Lissajous Type: (-17, 19, 3.2339924375189044, -12)
    (primitive) (-17, 19, (3.2339924375189044, '35/34 π'))
Lissajous Curve: sin(-34πt) + i sin(38πt+3.2339924375189044) (t : a real number)

 Slope 1/1  /  Level: 3
- - - - - - - - - - - - - - - - - - - - -
  ε': 011011010010010010110110100 1001001
  ε: +--+--+-++-++-++-+--+--+-++-++-++-
  W: ABABBBBABABBBBABABBABAABABBABAABABBABAABABBABABBBBABABBBBABABBABAABABBABAABABBAB
AABABB
  V: ABABBBBABABBBBABABBABAABABBABAABABBABAABABB
  Wbdpq: qpqpqbdbdbdbqpqpqbdbdbdb
  Vbdpq: qpqpqbdbdbdb
  PSL2: [[36, 259], [185, 1331]]
  PSL2/V: [[1, 7], [5, 36]]
  Dilatation: (1367 + sqrt(1868685)) / 2  (value=1366.999268470713)
  Dilatation/V: (37 + sqrt(1365)) / 2  (value=36.97295320191117)
- - - - - - - - - - - - - - - - - - - - -
  phi0: (3.2339924375189044, '35/34 π')
  st_ε': 011011010010010010110110100 1001001
  st_ε: -++-++-+--+--+--+-++-++-+--+--+--+
  st_W: ABBABBABBABBABBABABBAABBABABBAABBABABBAABBABABBABBABBABBABABBAABBABABBAABB
ABABBAABBAB
  st_Wbdpq: qpqpqbdbdbdbqpqpqbdbdbdb
```

```
  AB: ABABABABABABBABBABBABBABBABBABBABABABABABABABBABBABBABBABBABBABB
  R: ABBABBABBABBABBABABBAABBABABBAABBABABBAABBABABBABBABBABBABABBAABBABABBAABBABA
BBAABBAB
  Rbdpq: qpqpqbdbdbdbqpqpqbdbdbdb
  st_RL: R^5 L^7 R^5 L^7
- - - - - - - - - - - - - - - - - - - - -
  Christoffel Word: 01
  Palindromic Conjugate: 10
  δ': 10110111011011011
  δ'->ε': 011011010010010010110110100100l001
  r (4.11): (4, 3)
  V (4.11): bdbdbdbqpqpq
  R (4.11): bdbdbdbqpqpqbdbdbqpqpqpq

        >>> lis.set([7,-11,17/14*math.pi], silent=True)
        >>> lis.show()

Lissajous Type: (7, -11, 3.365992128846207, 6)
    (primitive) (7, -11, (-3.814791079359034, '-17/14 π'))
Lissajous Curve: sin(14πt) + i sin(-22πt+3.365992128846207) (t : a real number)

 Slope 1/3  /  Level: 1
- - - - - - - - - - - - - - - - - - - - -
  ε': 10110101011010
  ε: +-++-+-+-++-+-
  W: BABBABBABBABABBABABBABBABBABABBA
  V: BABBABBABBABABBA
  Wbdpq: pdbdpdpdbdpd
  Vbdpq: pdbdpd
  PSL2: [[116, -65], [-91, 51]]
  PSL2/V: [[9, -5], [-7, 4]]
  Dilatation: (167 + sqrt(27885)) / 2  (value=166.99401176132335)
  Dilatation/V: (13 + sqrt(165)) / 2  (value=12.922616289332565)
- - - - - - - - - - - - - - - - - - - - -
  phi0: (-3.814791079359034, '-17/14 π')
  st_ε': 10101101010110
  st_ε: +-+-++-+-+-++-
  st_W: BABBABABBABBABBABABBABABBABBABBA
  st_Wbdpq: pdpdbdpdpdbd
  AB: BABBABBABBABABBABABBABBABBABABBA
  R: BABBABABBABBABBABABBABABBABBABBA
  Rbdpq: pdpdbdpdpdbd
  st_RL: R^(-1) L^(-1) R^(-1) L^(-3) R^(-1) L^(-1) R^(-1) L^(-3)
- - - - - - - - - - - - - - - - - - - - -
  Christoffel Word: 0001
  Palindromic Conjugate: 1000
  δ': 1110111
  δ'->ε': 01011010101101
  r (4.11): (2, 1, 1, 1)
  V (4.11): dbdpdp
  R (4.11): dbdpdpdpdpqp

        >>> lis.set([-17,25,41/34*math.pi], silent=True)
        >>> lis.show()

Lissajous Type: (-17, 25, 3.788391141093574, -14)
    (primitive) (-17, 25, (3.788391141093574, '41/34 π'))
Lissajous Curve: sin(-34πt) + i sin(50πt+3.788391141093574) (t : a real number)

 Slope 3/5  /  Level: 1
- - - - - - - - - - - - - - - - - - - - -
  ε': 0101101010010100101011010100101001
  ε: +-+--+-+-++-+-++-+-+--+-+-++-+-++-
```

```
  W: ABABBABABBBBABABBABABBABAABABBABABBABAABABBABABBABABBBBABABBABABBABAABABBABABBAB
AABABB
  V: ABABBABABBBBABABBABABBABAABABBABABBABAABABB
  Wbdpq: qbqpqbqbdbqbdbqbqpqbqbdbqbdb
  Vbdpq: qbqpqbqbdbqbdb
  PSL2: [[12996, 49135], [23141, 87491]]
  PSL2/V: [[41, 155], [73, 276]]
  Dilatation: (100487 + sqrt(10097637165)) / 2  (value=100486.99999004847)
  Dilatation/V: (317 + sqrt(100485)) / 2  (value=316.9968453944747)
- - - - - - - - - - - - - - - - - - - - -
  phi0: (3.788391141093574, '41/34 π')
  st_ε': 0101101010010100101011010100101001
  st_ε: -+-++-+-+--+-+--+-+-++-+-+--+-+--+
  st_W: ABBABABBABBABBABABBABABBAABBABABBABABBAABBABABBABABBABBABABBABABBAABBABABB
ABABBAABBAB
  st_Wbdpq: qbqpqbqbdbqbdbqbqpqbqbdbqbdb
  AB: ABABBABABABABBABABBABBABBABABBABBABBABABBABABABABBABABBABBABBABABBABBABB
  R: ABBABABBABBABBABABBABABBAABBABABBABABBAABBABABBABABBABBABABBABABBAABBABABBABA
BBAABBAB
  Rbdpq: qbqpqbqbdbqbdbqbqpqbqbdbqbdb
  st_RL: R L R^3 L R L^3 R L^3 R L R^3 L R L^3 R L^3
- - - - - - - - - - - - - - - - - - - - -
  Christoffel Word: 00100101
  Palindromic Conjugate: 01001001
  δ': 11011111011110111
  δ'->ε': 0100101011010110101001010110101101
  r (4.11): (1, 2, 1, 1, 2, 1, 1, 2)
  V (4.11): bqpqbqbdbqbqpq
  R (4.11): bqpqbqbdbqbqpqbdbqbqpqbqbdbq
```

[３つのクラスの利用]　　**ThreeBraid** クラス、**LinearBraid** クラス、**LissajousBraid** クラスを利用して、表現形式の簡約化、表現形式の間の変換を計算してみます。ただしこれらは、**tb.tr.** 以下に定義された関数を使って計算できます。**ThreeBraid** クラスの **val()** メソッドを使います。AB 型表現形式で与えられた 3-braid（上端下端のあるもの）の Sigma 型表現形式で表すには、**val()** メソッドに引数 **'Sigma'** を指定します。

```
        >>> tb.ThreeBraid('ABBABABBAB').val('Sigma')

2 -1 2 -1
```

同じことを **tb.tr.toSigma()** 関数でも計算できます。

```
        >>> tb.tr.toSigma('ABBABABBAB')

2 -1 2 -1
```

val() メソッドは、**ThreeBraid** クラスの内部インスタンス変数 **'braid'** に格納された値を呼び出しているだけなので、引数には辞書型オブジェクト **'braid'** の任意のキーを指定できます。実際には７つの表現形式に関する次の７つ **'AB', 'Syzygy', 'Sigma', 'BDPQ'**（**'Frieze'**）, **'RL', 'ST', 'PSL2'** を指定してください。**tb.tr.** 以下にそれぞれの形式に変換する関数が定義されています。**tb.tr.toAB**, **tb.tr.toSyzygy**, **tb.tr.toSigma**, **tb,tr,toBDPQ**（**tb.tr.toFrieze**）, **tb.tr.toRL**, **tb.tr.toST**, **tb.tr.toPSL2** です。ただしこれらの関数では、それぞれの表現形式での簡約化は完了していません。簡約化のためには **tb.tr.reduce** 関数を用意しています。

```
        >>> tb.tr.toAB('BBABBBAABBAAABBAABBABAABABB')

BBABBBAABBAAABBAABBABAABABB     >>> tb.tr.reduce(tb.tr.toAB('BBABBBAABBAAABBAABBABAAB
ABB'))

BBABBABABBABB   >>> tb.ThreeBraid('BBABBBAABBAAABBAABBABAABABB').val('AB')

BBABBABABBABB
```

val() メソッドの第２引数に、show() メソッドのキーワード引数 brStyle で指定できる 'braid', 'closure', 'loop', loop3, 'loop6' を指定し、そのスタイルでの 3-braid の（簡約化された）表現形式を求めることができます。

```
>>> tb.ThreeBraid('BBABBBAABBAAABBAABBABAABABB').val('AB', 'closure')
```

ABBABABBAB

LinearBraid クラス、LissajousBraid クラスにも val() メソッドが定義されています。linear parameter で定義された linear 3-braid や、Lissajous type で定義された Lissajous 3-braid を７つの表現形式で表すことができます。ただし、shape sphere 上の閉路の３分の１（loop3）を規定値としています。

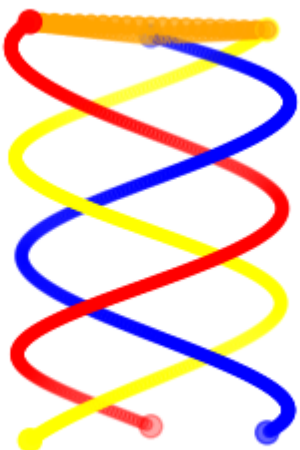
[Plot] グラフ表示のためのライブラリなど（**matplotlib, numpy, ipywidgets, IPython, mpl_toolkits**）と、jupyter notebook などのグラフ表示可能な端末で、グラフ表示に必要な拡張機能がインストールされていれば、３つのクラス（ThreeBraid, LinearBraid, LissajousBraid）で視覚化メソッド plot() が利用できます。plot() メソッドは、３次元空間内における 3-braid の３次元像（'braid'）、3-braid の断面における３点の動きの平面像（'orbit', 'curve'）（3-braid の３次元像を端点側から見たもの）、3-braid に対応する shape sphere 上の道（'pathSS', 'loopSS'）と平面への投影像（'path', 'loop'）の４種類です。plot() メソッドの第１引数に 'braid', 'curve', 'pathSS'（'loopSS'）, 'path'（'loop'）を指定することで、目的の像が描かれます。LinearBraid クラスと LissajousBraid クラスで扱う 3-braid は、Lissajous 曲線上の３点の動きで生成される 3-braid なので、Lissajous 曲線を使って描くことができます。ThreeBraid クラスで扱われる任意の 3-braid には、そのような枠となる曲線が与えられてはいないので、LinearBraid クラスや LissajousBraid クラスとは別の視覚化手続きを必要とします。最初に、Lissajous 曲線を使った視覚化メソッドの使い方を説明し、その後で ThreeBraid クラスの視覚化メソッドの実現手順などについて説明します。

[LissajousBraid クラスの視覚化メソッド] LissajousBraid クラスで説明します。LinearBraid クラスでも全く同じです。Lissajous type を与えて LissajousBraid クラス型オブジェクト変数を用意しておきます。

```
>>> lis = tb.LissajousBraid([4,-5, 1], silent=True)
```

Lissajous 3-braid は、枠となる Lissajous 曲線が具体的に与えられています。平面に描かれる Lissajous 曲線の媒介変数を第３の座標にとることで３次元空間内の曲線になります。媒介変数を 1/3 ずつずらして３本の曲線を描いたものが、Lissajous 3-braid の３次元像です。plot('braid') で描かれる３次元像は、媒介変数を上から下に動かしています。上が小さい値で下に行くほどが大きい値になります。

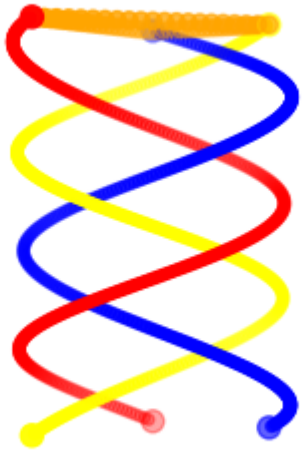
```
>>> lis.plot('braid')
```



右側に３次元像を動かすためのスライダーがあります。「仰角」のスライダーを動かして上方もしくは下方から眺めることができます。「視線」のすランダーを動かして像を回転させることができます。初期設定では、「仰角」を「０」に、「視線」を「−９０」にしています。

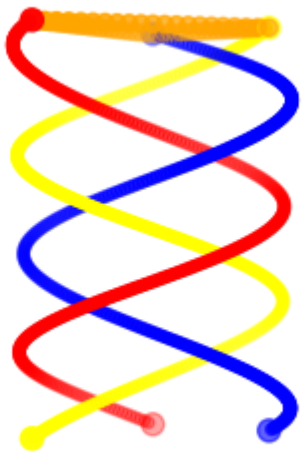
3-braid を生成する３点の動きを 3-braid 上に描くことができます。キーワード引数 **triangles** に、真偽値 **True**を指定してください。デフォルト値は **False** です。右側の「３動点」のスライダーで動かすことができます。

```
>>> lis.plot('braid', triangles=True)
```

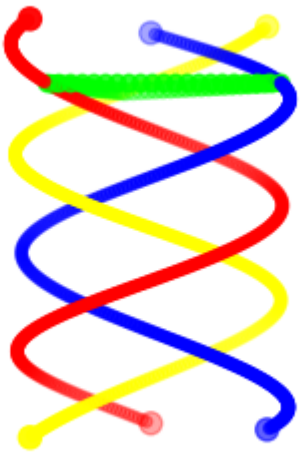


右側のスライダーなどが不要な場合は、**snapshot=True** と指定してください。キーワード引数 **snapshot** に真偽値でなく実数値を指定すると、指定された値における３動点の配置を描きます。

```
>>> lis.plot('braid', snapshot=True)
```

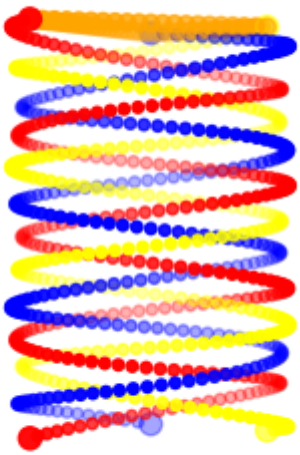


```
>>> lis.plot('braid', snapshot=0.05)
```

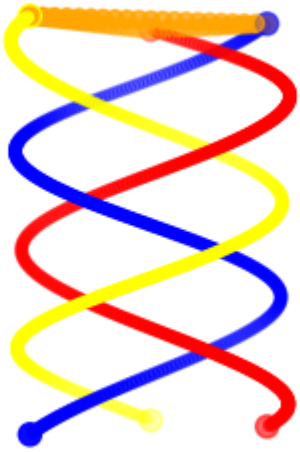


初期設定では、**LissajousBraid** クラスの描画像は、媒介変数 t の動く範囲を 0 ~ 1/3 にしています。周期の 1/3 だけを描くようにしています。描画範囲（媒介変数 t の動く範囲）を任意にしていすることができます。キーワード引数 **T** に範囲をしていしてください。例えば、**T=(0,1)** とすると、一周期分になります。また、**T=(1/3,2/3)**, **T=(2/3,1)** と指定しすることで、当初描かれた像の残りの 2/3 の部分を 1/3 ごとに描くことができます。また、**T=(-0.05,0.05)** と指定することで、t=0 の近くでの様子をみることもできます。

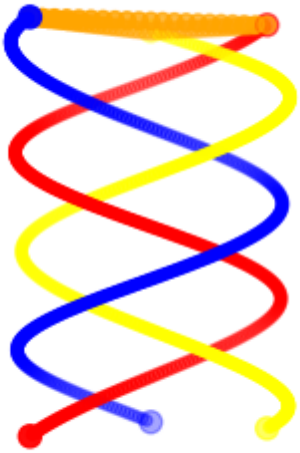
```
>>> lis.plot('braid', T=(0,1))
```



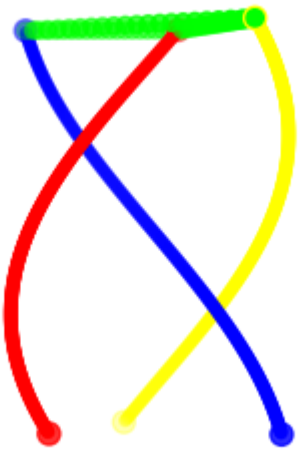
```
>>> lis.plot('braid', T=(1/3,2/3))
```



```
>>> lis.plot('braid', T=(2/3,1))
```

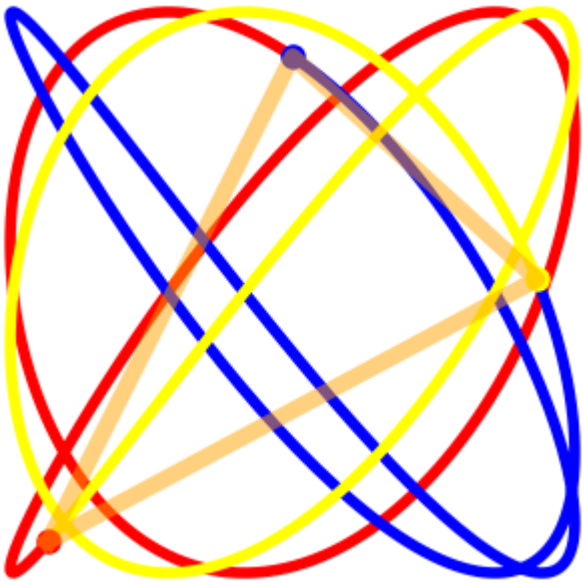


```
>>> lis.plot('braid', T=(-0.05,0.05))
```



３動点の動きの平面像（軌道）を描くには、**plot('orbit')** とします。**LissajousBraid** クラスの場合は Lissajous 曲線を描くことになります。媒介変数の範囲によっては、Lissajous 曲線の一部しか描かれない場合もあります。３動点の動きを一つずつ重ねて描くので、後から描いた点が、先に描かれた点の上に描かれ、隠されてしまうことがあります。媒介変数の動く範囲が 1/3 周期分のときは、３点（３色）で Lissajous 曲線が描かれます。1/3 周期より小さいときは一部が描かれず、1/3 周期より大きいときは一部が重なり先に描かれた点の軌道が見えなくなります。

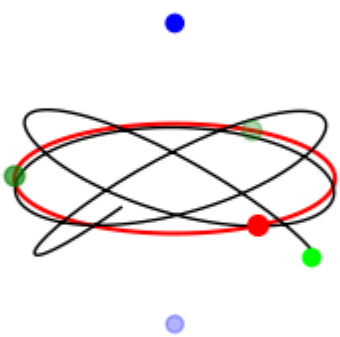
```
>>> lis.plot('orbit')
```



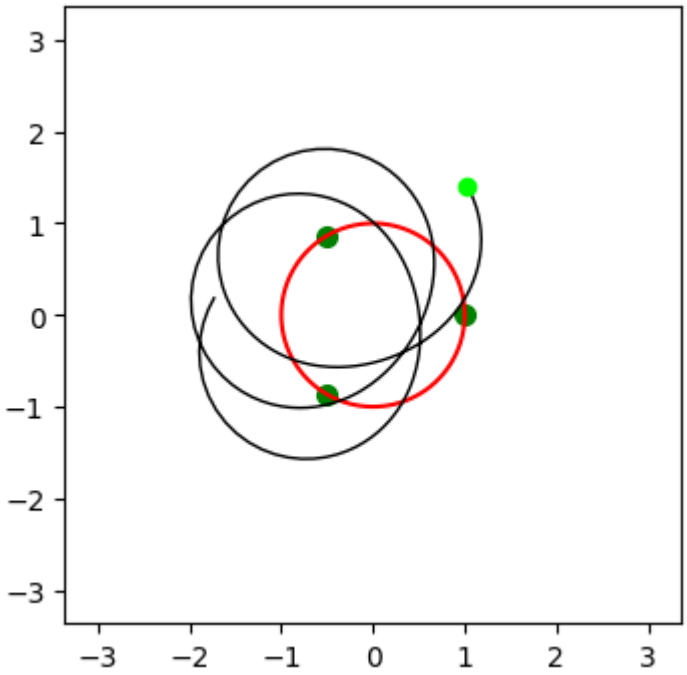
plot('orbit') の場合も、３動点を描きたい場合は、**triangles=True** と指定してください。右側のスライダーなどがいらないときはキーワード引数 **snapshot** を指定し、媒介変数 t の変動範囲を変更するときはキーワード引数 **T** に適当な範囲をしていしてください。これらの使い方は **plot('braid')** と同じです。

plot('pathSS') で、Lissajous 3-braid に対応する shape sphere（３次元空間内の単位球面）上の閉路（Lissajous loop）を描くことができます。shape sphere の北極から shape sphere の中心と赤道を通る平面への射影により、shape sphere 上の道を平面に投影することできます。**plot('path')** で、Lissajous loop の平面上への射影像を描くことができます。shape sphere 上の北極と南極は、正三角形に対応します。Lissajous 曲線の実部と虚部に異なる重みをつけて、３動点の配置が正三角形にならないようにすれば、shape sphere に描かれた Lissajous loop を平面の有界な範囲に描くことができます。初期値は、(実軸方向の変動範囲)／(虚軸方向の変動範囲)＝３ にしています。

```
>>> lis.plot('pathSS')
```



```
>>> lis.plot('path')
```


3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

これらに対しても、キーワード引数 **triangles**, **snapshot**, **T** を利用できます。ただし shape sphere 上の道において、「３動点」として道の上の点を指すのは少し不自然かもしれません。

LissajousBraid クラスと **LinearBraid** クラスの視覚化メソッド（**plot('braid')** , **plot('orbit')**, **plot('pathSS')**, **plot('path')**）は、予め与えられた Lissajous type に対して、Lissajous 3-braid を視覚化するものですが、Lissajous type をインタラクティブに指定して、Lissajous 曲線や Lissajous 3-braid, Lissajous loop を描くために、**plot_Lissajous** 関数を用意しています。図の右側のスライダーを使って、Lissajous type を変更することができます。plot_Lissajous 関数は引数を一つもちます。３次元空間内に Lissajous 3-braid を描くための **plot_Lissajou('braid')**、平面上に Lissajous 曲線を描くための **plot_Lissajou('orbit')**、Lissajous loop を Shape sphere 上に描くための **plot_Lissajous('pathSS')** と **plot_Lissajous('path')**）です。

```
>>> tb.plot_Lissajous('braid')
```

（図は略します）

```
>>> tb.plot_Lissajous('orbit')
```

（図は略します）

```
>>> tb.plot_Lissajous('pathSS')
```

（図は略します）

```
>>> tb.plot_Lissajous('path')
```

（図は略します）

tb.plot_Lissajous 関数に対しても、キーワード引数 **triangles**, **T** は機能しますが、**snapshot** は使えません。

キーワード引数 **new** により、Lissajous 曲線の定義式が [KNO2] に従うかどうかを指定できます。デフォルト値 **new=True** のとき [KNO2] に従い **LissajousBraid** クラスで使っている Lissajous 曲線の定義式の Lissajous type になります。**new=False** のときは **LinearBraid** クラスで使っている Lissajous 曲線の定義式の Lissajous type になります。

[一般 3-braid の視覚化]　linear 3-braid や Lissajous 3-braid は、Lissajous 曲線上を動く３点を使って描くことができます。Lissajous 曲線のような整った枠の与えられていない一般の 3-braid に対する詳細は pdf ファイル（ThreebraidClassModule.pdf）をご覧ください。

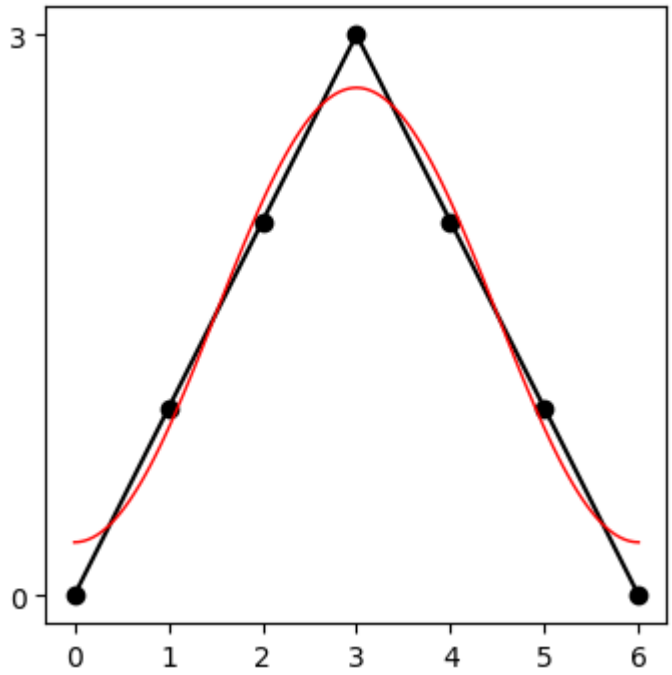
具体的に描いてみます。**ThreeBraid** クラス型オブジェクト変数を用意します。

```
>>> br = tb.ThreeBraid('ABABBABABABBAB')
>>> br.show('braid', ['AB', 'Syzygy', 'Sigma'])
```

```
Braid Style: braid
  AB: ABABBABABABBAB
  Syzygy: 0120210
  Sigma: σ1^-1 σ2 σ1^-2 σ2 σ1^-1
```

shape sphere 上の道 γ(t) の偏角 θ(t) は **plot('Fexp')** で描くことができます。syzygy 列により指定されたチェックポイントを通る経路となっていることを確認するためのものです。必要とする場面は多くはないと思います。返り値として、Fourier 係数のリストを持ちます。

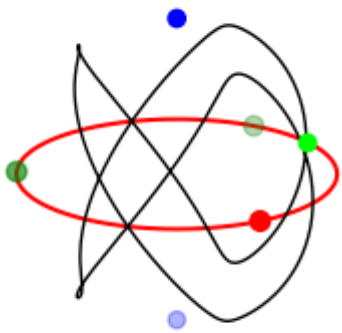
```
>>> br.plot('Fexp')
```



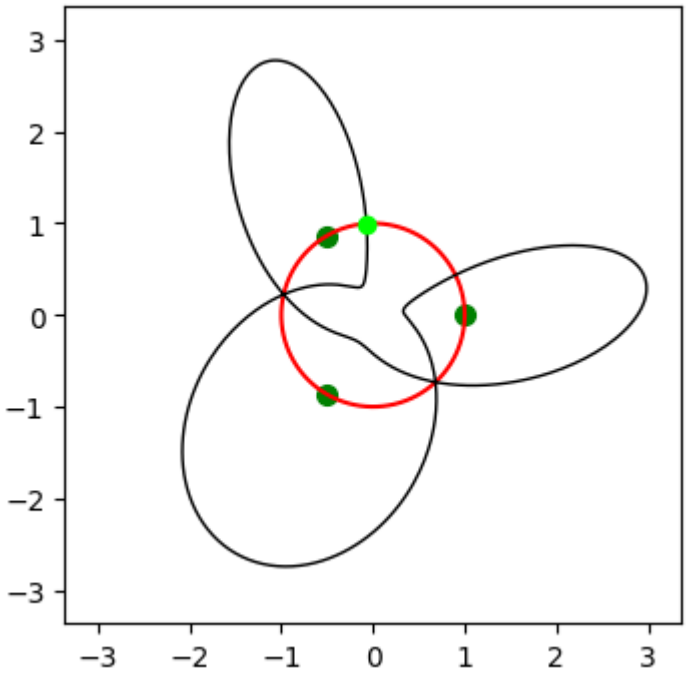

```
[('0', '2', 1.5, ('0', '-3', '-1/4', '6', '6')), [0.0, -1.21585, 0.28414]]
```

目的の 3-braid の３次元像を描くために必要な shape sphere 上の道は **plot('pathSS')** もしくは **plot('path')** で描くことができます。前者は３次元に置かれた球面上の道で、後者は北極から平面への投影像です。

```
>>> br.plot('pathSS')
```



```
>>> br.plot('Path')
```



3-braid の３次元像は **plot('braid')** で描くことができます。

```
>>> br.plot('braid')
```

最後に紹介するのは、平面上での３動点の動きの軌跡を描く **plot('orbit')** です。単位正方形内に収まっていた Lissajous 曲線とは異なり、楕円（長径２、短径１）内の曲線になります。縦横の表示比率はグラフが描画領域に収まるように自動調整されているため、実際の図形とは異なります。表示比率を揃えたい場合は、引数に **aspect='equal'** を指定してください。

```
>>> br.plot('orbit')
```

```
>>> br.plot('orbit', aspect='equal')
```

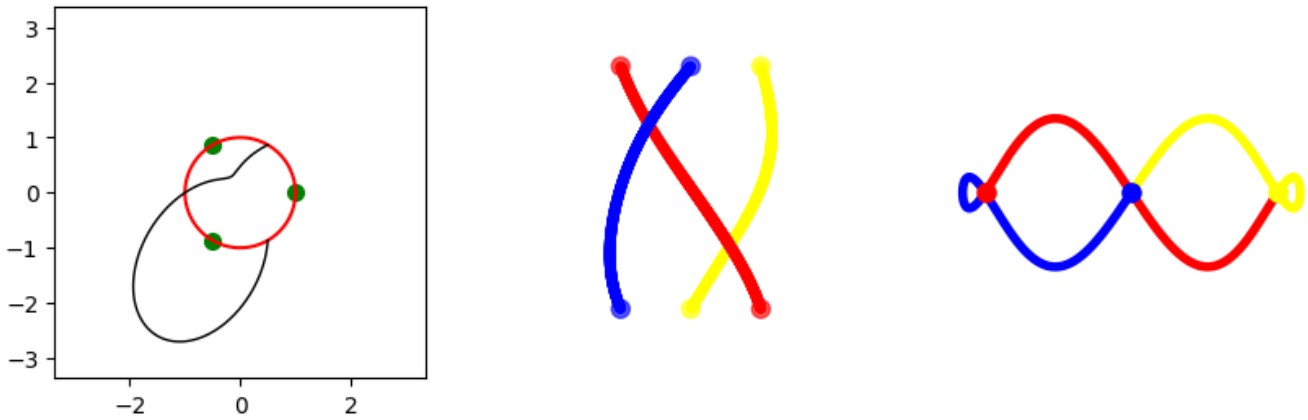
ここまで描いたものは、ThreeBraid クラスの 'braid'（上端下端のある braid）です。対応する shape sphere 上の道の始点終点を、その点の属する区画内の赤道（syzygy の状態）まで延長し、3-braid の３次元像は３本の紐の３つの端点が一直線上に並んでいます。'closure' は 'braid' の上端と下端をつないだもので、'loop' は同じ 'braid' を幾つかつないで対応する shape sphere 上の道が閉路となるものでした。'closure' にしろ 'loop' にしろ表現形式を巡回語とみることで、簡約化がすすむことがあります。'braid' としての描画像と 'closure' や 'loop' としての描画像は異なります。上で描いたものは、'braid' です。'closure' や 'loop' として描くには、**plot()** メソッドの第２引数に **'braid', 'closure', 'loop'** を指定してください。第２引数を省略した場合は、**'braid'** が補われます。**startArea='0+ '** で syzygy 列 **'012'** で与えた 3-braid について、'braid', 'closure', 'loop' での違いをみてみましょう。まずは、**ThreeBraid** クラス型変数を用意します。

```
>>> br = tb.ThreeBraid('012', startArea='0+ ')
>>> br.show(['braid', 'closure', 'loop'], ['AB', 'Syzygy', 'Sigma'])
```

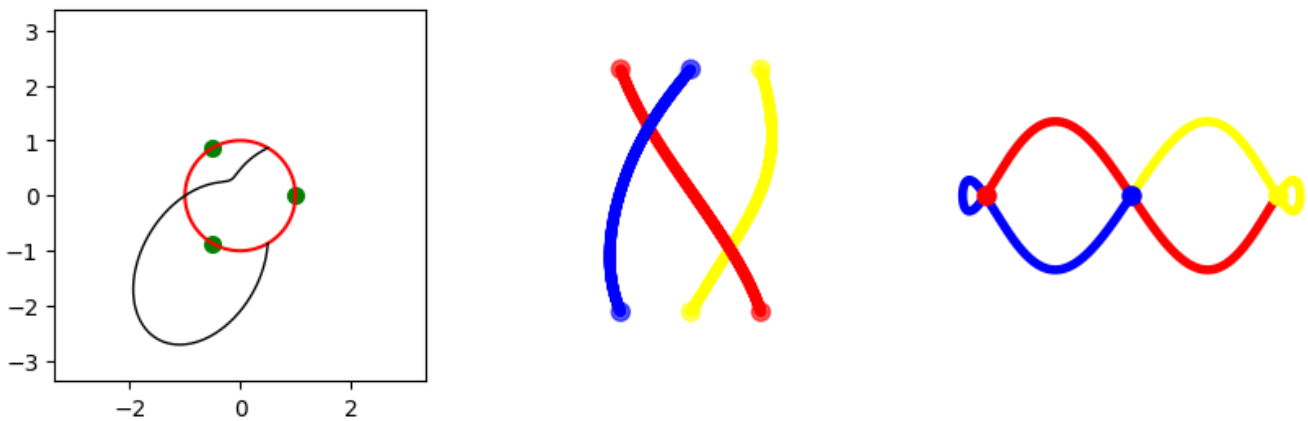
```
Braid Style: braid
  AB: ABABB
  Syzygy: 012
  Sigma: σ1^-1 σ2
Braid Style: circular word
  AB: ABABB
  Syzygy: 01
  Sigma: σ1^-1 σ2
Braid Style: loop on shape sphere
  AB: ABABBABABBABABB
  Syzygy: 012012
  Sigma: σ1^-1 σ2 σ1^-1 σ2 σ1^-1 σ2
```

３つのスタイル 'braid', 'closure', 'loop' のそれぞれについて、shape sphere 上の道、braid の３次元像、３動点の軌跡を描いてみます。実は、**plot('all')** メソッドで、静止像ではありますが、その３つの図を一度に描くことができます。

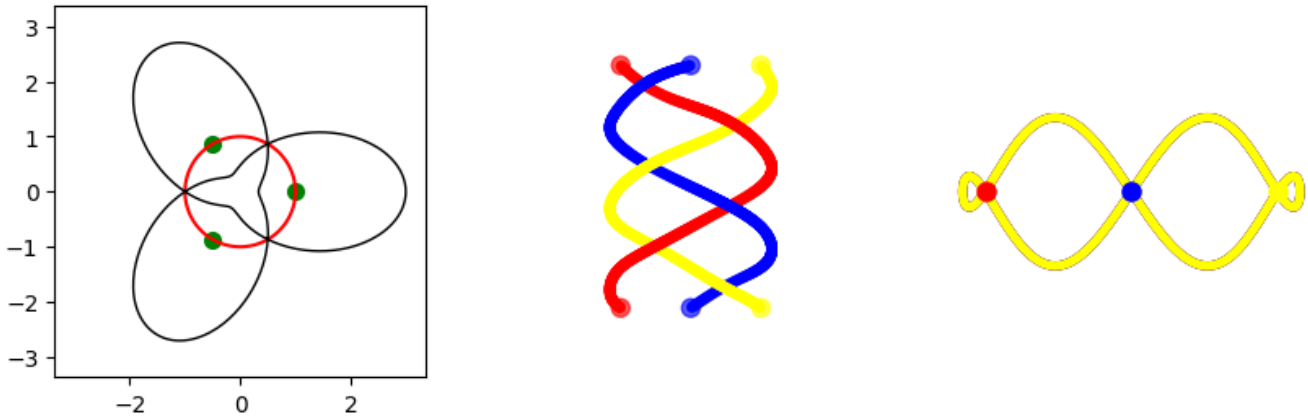
```
>>> br.plot('all, 'braid')
```




```
>>> br.plot('all', 'closure')
```

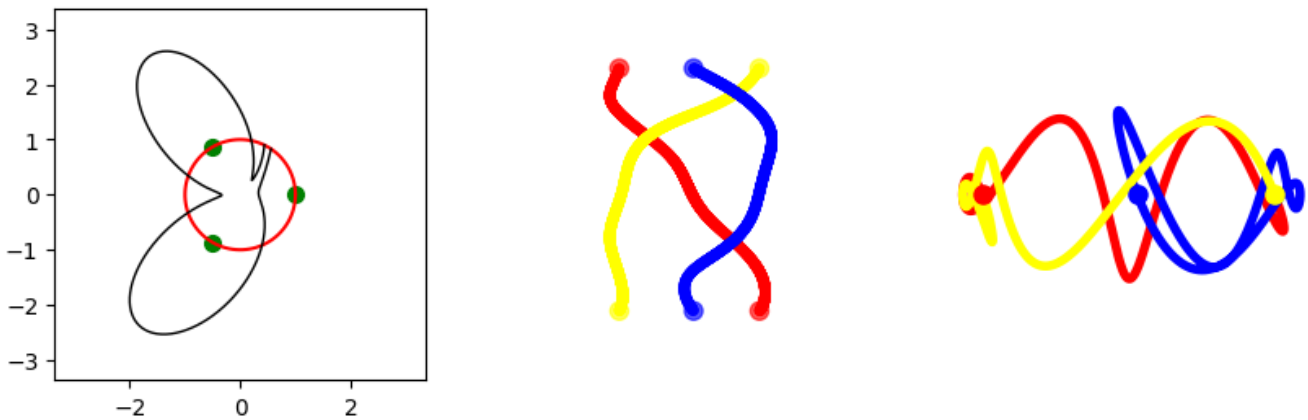


```
>>> br.plot('all', 'loop')
```

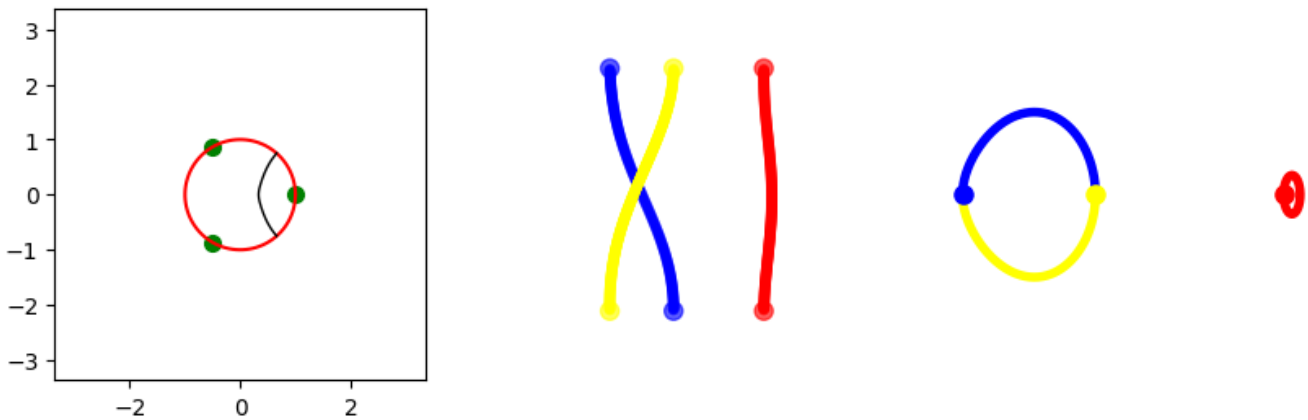


もう少し、複雑な例でみてみましょう。

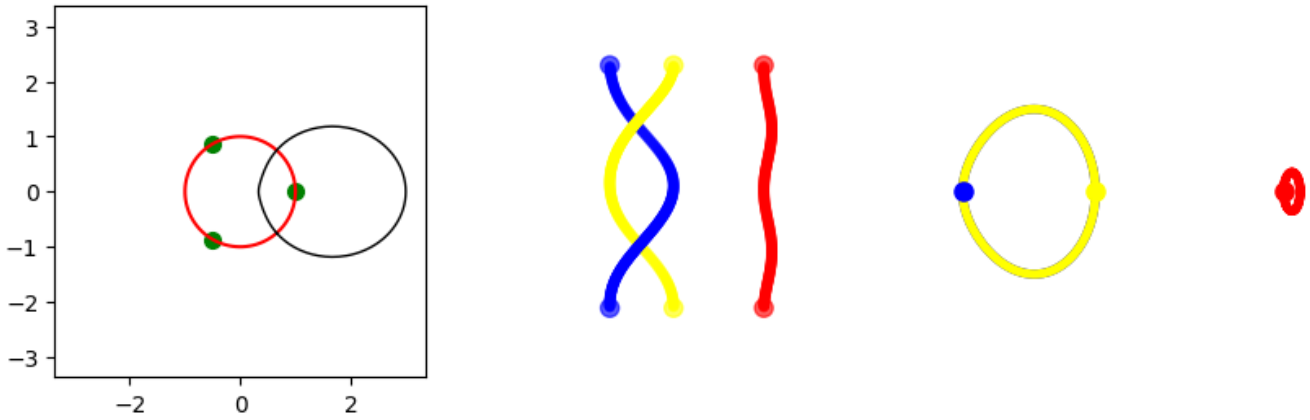
```
>>> br = tb.ThreeBraid('ABBABBA', startArea='0+ ')
>>> br.show(['braid', 'closure', 'loop'], ['AB', 'Syzygy', 'Sigma'])

Braid Style: braid
  AB: ABBABBA
  Syzygy: 0200
  Sigma: σ2 σ1^-1 σ2^-1
Braid Style: circular word
  AB: AB
  Syzygy: 2
  Sigma: σ1^-1
Braid Style: loop on shape sphere
  AB: ABAB
  Syzygy: 21
  Sigma: σ1^-2
>>> br.plot('all, 'braid')
```



```
>>> br.plot('all', 'closure')
```



```
>>> br.plot('all', 'loop')
```

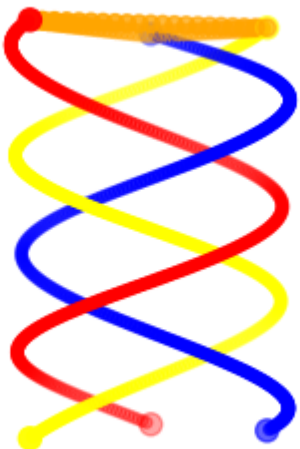


ThreeBraidClass モジュールの一般 3-braid の視覚化メソッドに使われる syzygy 列は、**ThreeBraid** クラスで計算された、簡約化された syzygy 列ではありません。**plot('braid')** での３次元像が最も見やすい形になるものを選んでいます。具体的には、簡約化された Sigma 型表現形式に沿った形に描けるように選んでいます。syzygy 列としては簡約化されていない冗長な列になっています。

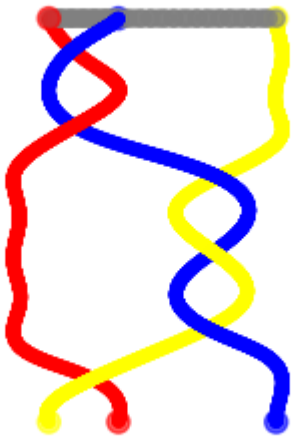
[3-braid の視覚化手続きを使って Lissajous 3-braid を描く] Lissajous 3-braid は Lissajous 曲線という枠を使って、視覚化していました。Lissajous 3-braid もまた 3-braid なので、3-braid の視覚化手続きに従って描くことができます。Lissajous 3-braid も Linear 3-braid も対応する shape sphere 上の道は閉路 'loop' になるので、ThreeBraid クラスで 'loop' を指定した描画像になります。**LissajousBraid** クラスと **LinearBraid** クラスの **plot('braid')** メソッド、**plot('curve')** メソッド、**plot('loopSS')** メソッド、**plot('loop')** メソッドのキーワード引数 **TB** に真偽値 True を割り当てることで、それぞれ ThreeBraid クラスの **plot('braid')** メソッド、**plot('curve')** メソッド、**plot('pathSS')** メソッド、**plot('path')** メソッドが使われます。Lissajous 曲線の枠をもとにした、**LissajousBraid** クラスの **plot('braid')** メソッドと、ThreeBraid クラスの **plot('braid')** メソッドで描かれる３次元像を比較してみましょう。

```
>>> lis = tb.LissajousBraid([4,-5, 1], silent=True)
>>> lis.plot('braid')

>>> lis.lb.show('loop3', ['AB', 'Syzygy', 'Sigma'])
```

```
Braid Style: loop/3 on shape sphere
  AB: ABABABBABBABBAB
  Syzygy: 010202
  Sigma: σ1^-2 σ2^3 σ1^-1
```



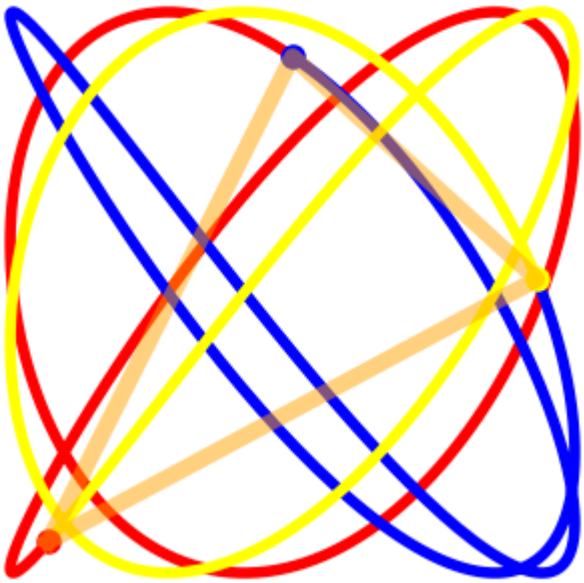
```
>>> lis.plot('braid', TB=True)
```



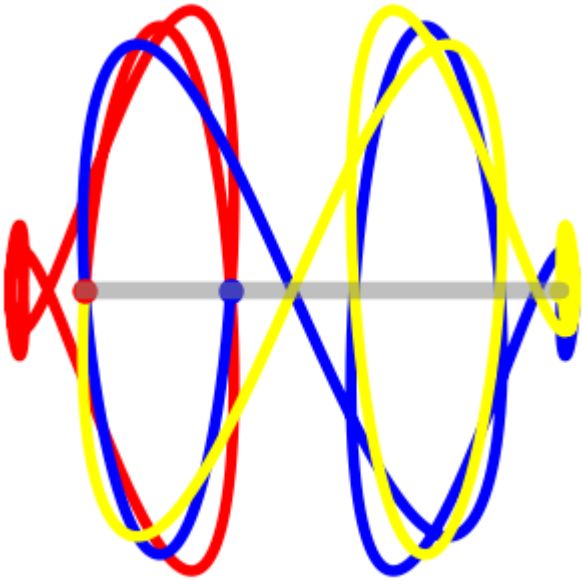
Lissajous 曲線上の３動点の動きを使って描いた Lissajous 3-braid の場合、Lissajous 曲線に従って３動点が全体として回転します。3-braid としては、３動点の絡み方に関心がありますが、３動点全体の回転に目立ってしまっています。一方、一般 3-braid の描画手続きでは、描画の第一歩の shape sphere 上の道を homotopy 同値類の中で適切なものを作るために、Syzygy 列を簡約でないやや冗長なものの中から上手く選びました。Syzygy のタイミングで３点の並びが一定の向きに揃うようにできたので、絡み具合を見えやすい図形を描くことができました。実際 3-braid 描画て手続きを利用したものでは、Sigma 表現形式そのままの形で絡まった 3-braid が描かれています。

両者の違いを見るために、３頂点の軌跡、shape sphere 上の道を比べてみます。**plot('curve')** メソッドでの頂点の軌跡を比べてみます。

```
>>> lis.plot('curve')
```

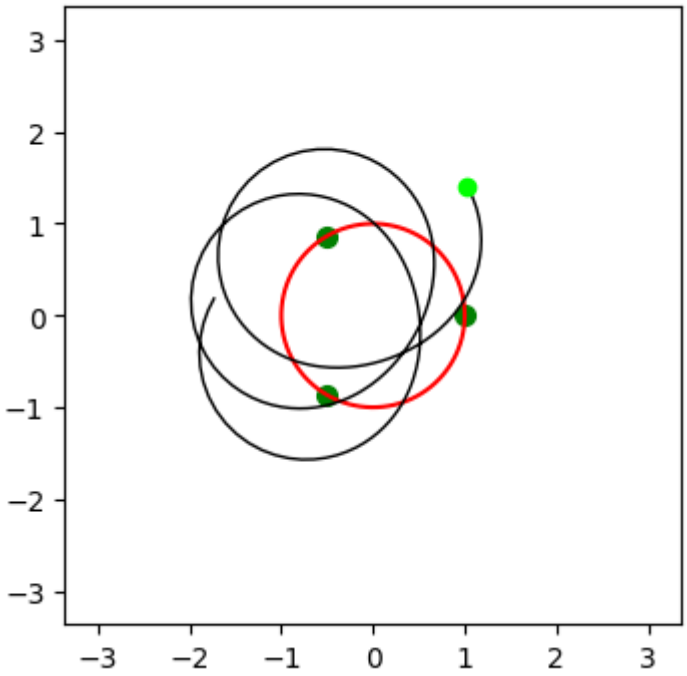


```
>>> lis.plot('curve', TB=True)
```

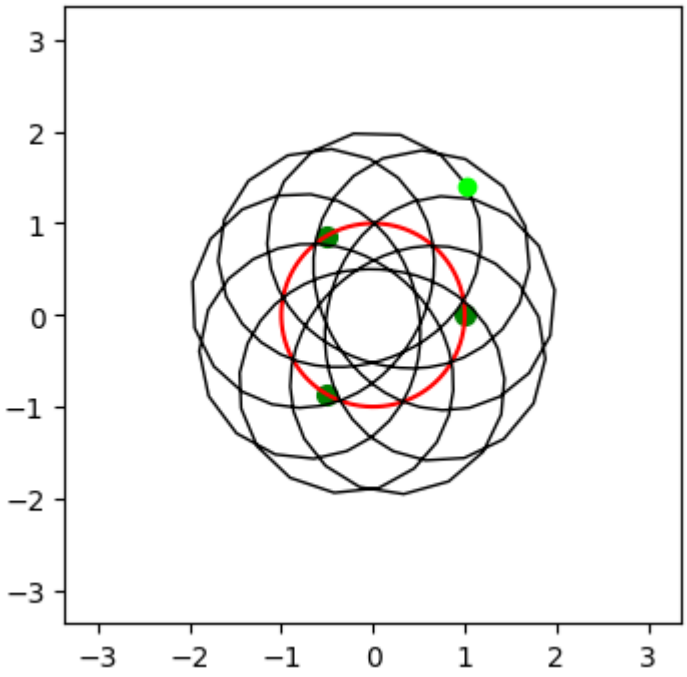


Shape sphere 上の閉路についても比べてみます。初期設定では、閉路の 1/3 を表示するようにしています。パラメータ t の変動域を 0≦t≦1/3 としています。全体を表示させたい場合は、キーワード引数 **T** にパラメータ t の変動域を指定してください。例えば、全体像を描くために 0≦t≦1 とする場合は、**T=(0,1)** としてください。あるいは、t=0 の近くのでの振る舞いを調べるために −0.1≦t≦0.1 とするには、**T=(−0.1,0.1)** と指定してください。この指定方法は、視覚化関係のメソッドで共通です。

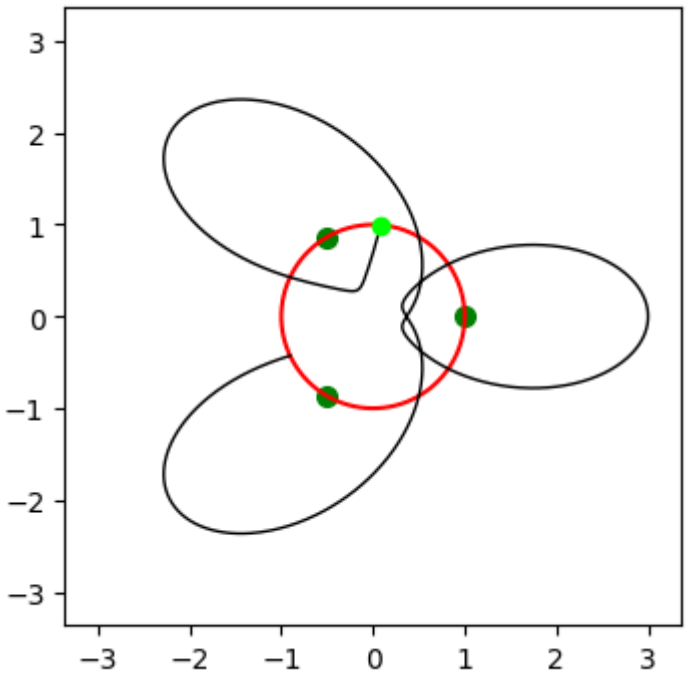
```
>>> lis.plot('loop')
```



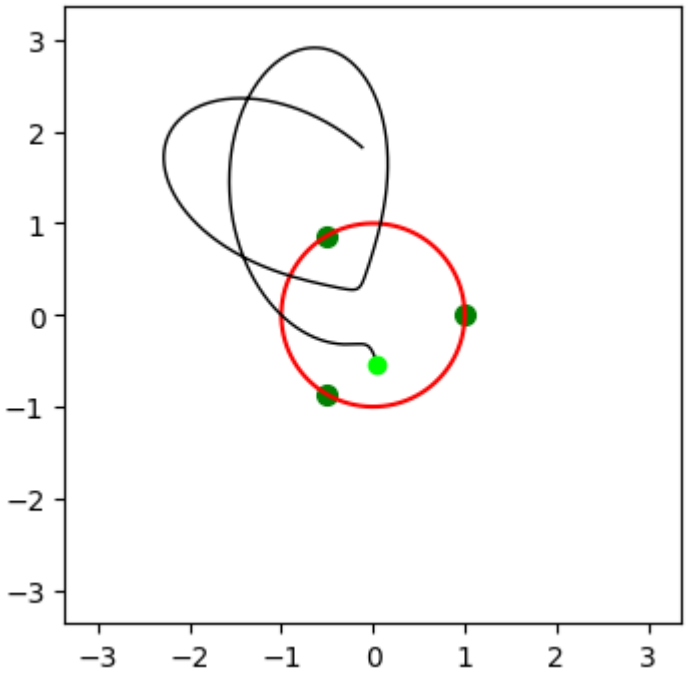

```
>>> lis.plot('loop', T=(0,1))
```



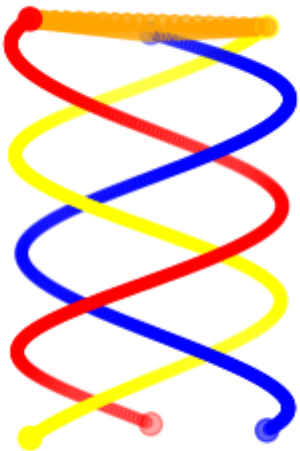

```
>>> lis.plot('loop', TB=True)
```



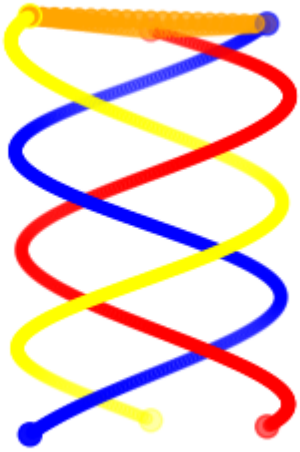

```
>>> lis.plot('loop', TB=True, T=(-0.1,0.1)
```



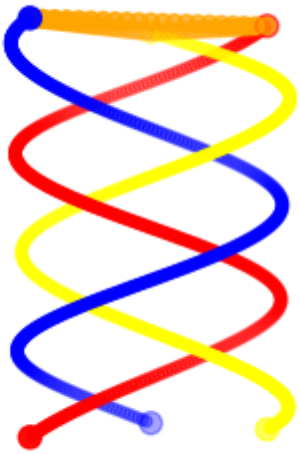

```
>>> lis.plot('braid')
```



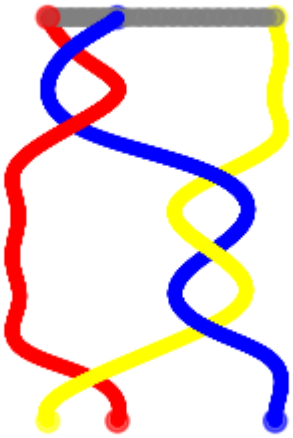
```
>>> lis.plot('braid', T=(1/3,2/3))
```



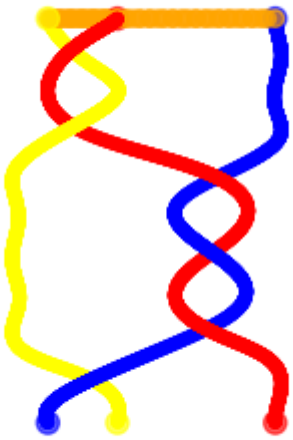
```
>>> lis.plot('braid', T=(2/3,1))
```



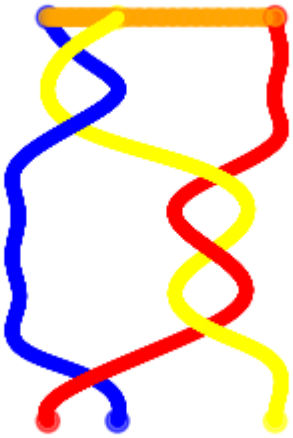
```
>>> lis.plot('braid', TB=True)
```



```
>>> lis.plot('braid', TB=True, T=(1/3,2/3))
```

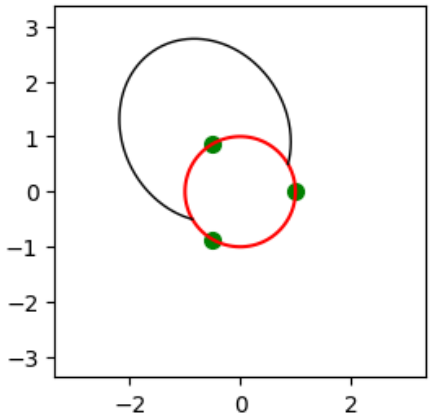
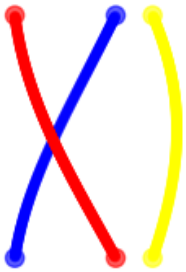
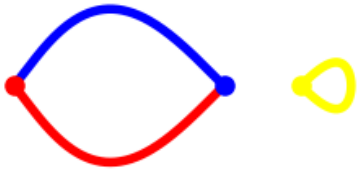


```
>>> lis.plot('braid', TB=True, T=(2/3,1))
```

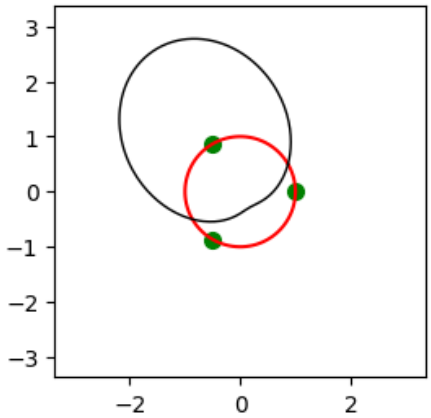


[視覚化手続きのみの利用]　ThreeBraid クラス型オブジェクト変数を用意することなく、3-braid の視覚化手続きのみを利用するために、**tb.plot_ThreeBraid 関数**を用意しています。使い方は基本的に **tb.plot_LissajousBraid 関数**と同じですが、今のところ、3-braid をインタラクティブに変更するためのインターフェイスは備えられていません。また、3-braid の３つのスタイルのうち、上端下端のある 'braid' の描画のみで、他の 'closure', 'loop' には対応していませんが、描画手続に Fourier 展開を使っているので、shape sphere 上の道が自然に拡張され、パラメータの範囲を適切に延長することで 'closure' や 'loop' に対応する描画像をえることができます。ただし１点のみ注意が必要なところがあります。それは、描きたい 3-braid を syzygy 列で与えた場合です。**ThreeBraid** クラスで描画像をつくるためには、まず AB 型表現形式に変換し、Sigma 型表現形式をへて、描画に適切な syzygy を与えています。その過程で適宜簡約化が行われ、与えられたそのままの形ではなく、3-braid として直感的にわかりやすい形に変換されています。ところが syzygy 列で 3-braid を与えるのは、shape sphere 上の道を指定し、その道に対応する 3-braid を考えたい場合です。指定した syzygy 列の指示通りの道を描けばいいのですが、その前後、syzygy 列の定義されていない前後の道をどう考えるのが自然であるかで、２通りの考え方があります。１つ目は syzygy 列により定められた 3-braid をつなぐことで、２つ目は shape sphere 上の道を syzygy 列の指示（の主旨）に従って延長することです。これらは同じではありません。shape sphere 上の道の始点と終点の向きが異なるとき、終点を延長した先が逆向きになります。具体的に説明します。syzygy 列 '01'、始点 '0+ ' で与えられた 3-braid で説明します。この 3-braid は AB 型表現形式で 'AB' で表されます。２つつなぐと 'ABAB' で、syzygy 列は '010' になります。syzygy 列 '01' を単純に２つつなぐと '0101' ですが、'01' は '0' が始点で '1' が終点で、この終点で次の列をつながねばなりません。そこで '01' を、偏角が大きくなる方に１つ隣の区画にうつること、と考えれば、始点が '1' の場合は、'12' になります。'01' と '12' を '1' のところでつなぐので、'012' になります。AB 型表現形式で表すと、'ABABB' になります。3-braid の延長は同じ 3-braid をつなぐことの方が自然だと思うので、**tb.plot_ThreeBraid** 関数でもその様にしています。ただし、syzygy 列の指示に主眼をおいて延長を考えることもできるように、**tb.plot_ThreeBraid** 関数に **connectSZ** を用意しています。syzygy 列で与えられた 3-braid について、**connectSZ=True** と指定することで、syzygy 列に従って延長されます。この場合、shape sphere 上の閉路になるために、最大で６倍しなければならない場合があります。Lissajous type が (m,n,*) (m≡n≡1 (3)) の Lissajous 3-braid は、m と n が共に奇数のとき、６分割されます。syzygy 列に限って言えば、syzygy 列の長さが 6*((m-n)/3) で、m,n に依らず平行な６つのブロックに分けられます。3-braid を切り分けるとき、syzygy の状態に切り目を入れるので、syzygy 列を文字列として分けた後で、終点を補う必要があります。syzygy の定める shape sphere 上の道の始点と終点での道の向きは、始点終点のある syzygy 列の文字数が奇数の場合は同じ向きで、偶数の場合は逆向きになります。上記例 '01' は始点終点があり、文字数が偶数なので、逆向きになります。実際、上向き '0+ ' で北半球に向けて出発し、北半球から赤道の区画 '1' に向かって、下向きに進み、終点は下向き'1-' です。この場合、syzygy 列での延長は、AB 列での延長と異なり、もとの 3-braid の鏡映をつなぐことになります。m と n の一方が偶数のとき、６分割された syzygy 列に対応する道の始点と終点の向きが異なり、m と n が共に奇数のとき、６分割された syzygy 列に対応する道の始点と終点の向きが一致します。Lissajous 3-braid を６つに分割し、それを「延長」して元の Lissajous 3-braid を描き出すためには、3-braid をつなぐのではなく、syzygy 列をつなぐ方が自然であると言えます。

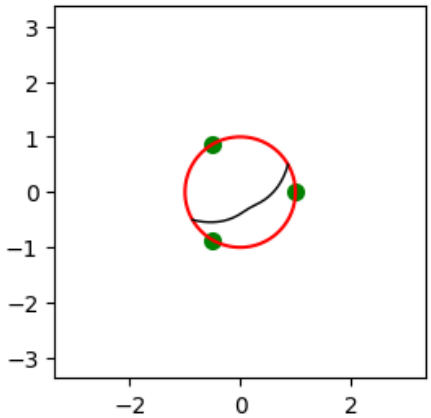
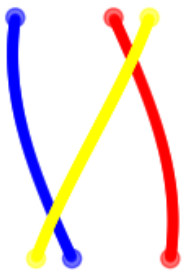
```
>>> tb.plot_ThreeBraid('01', 'all')
```

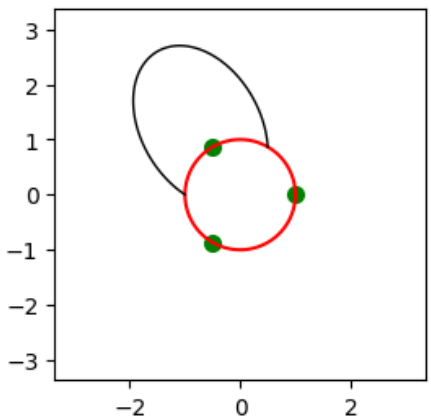
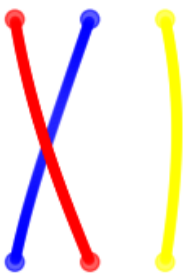
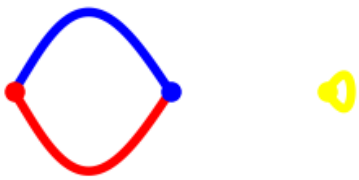
パラメータの区間を倍にしてみます。　　`>>> tb.plot_ThreeBraid('01', 'all', T=(0,1))`



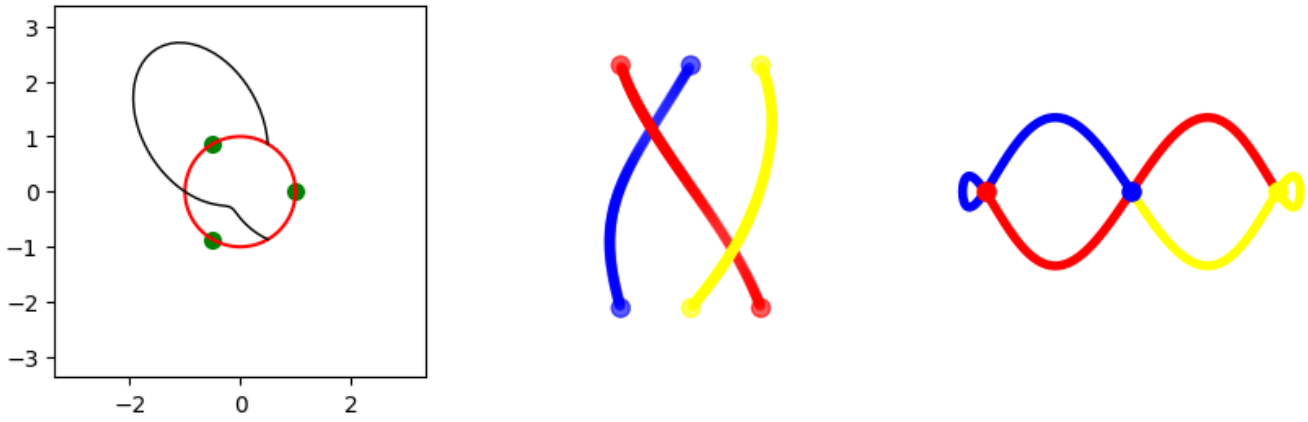
延長した部分だけを描くと。　`>>> tb.plot_ThreeBraid('01', 'all', T=(1/2,1))`

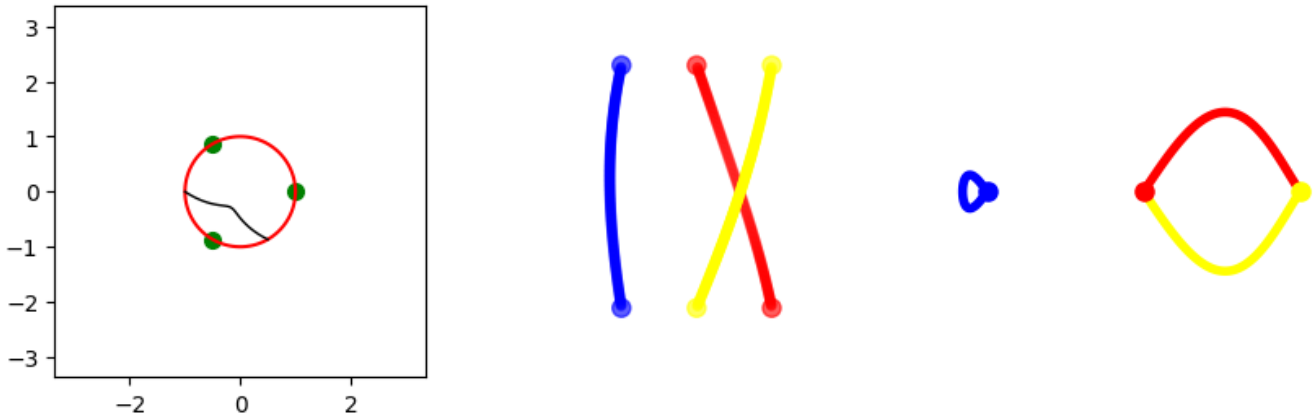

syzygy 列の延長を使って描くと。　　`>>> tb.plot_ThreeBraid('01', 'all', connectSZ=True)`

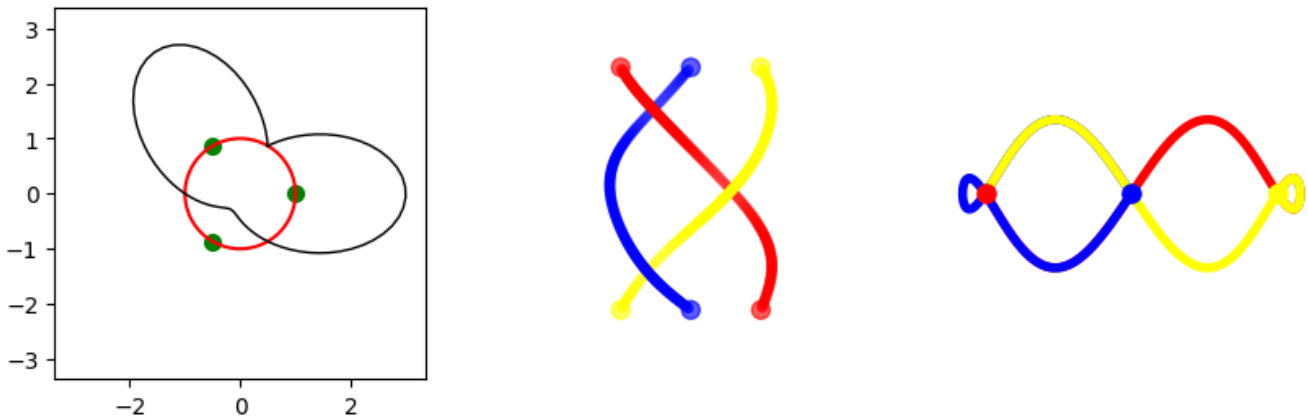
パラメータの区間を倍に伸ばすと。　　`>>> tb.plot_ThreeBraid('01', 'all', connectSZ=True, T=(0,2))`



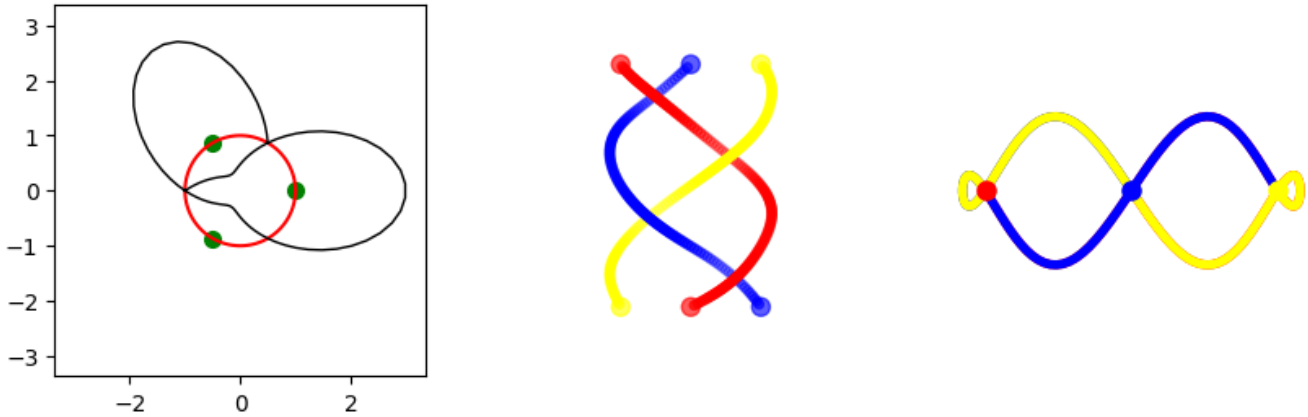
延長された部分は　>>> **tb.plot_ThreeBraid('01', 'all', connectSZ=True, T=(1,2))**



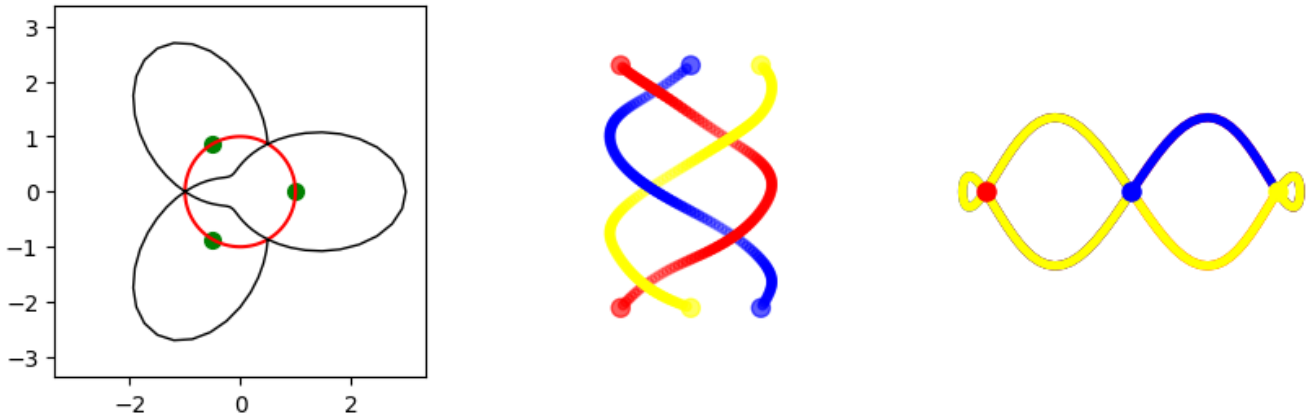
connectSZ=True では、'01' が '012' に延長されました。さらにその先は、'012012012...' と延長されることになります。パラメータの範囲を広げていきます。　>>> **tb.plot_ThreeBraid('01', 'all', connectSZ=True, T=(0,3))**



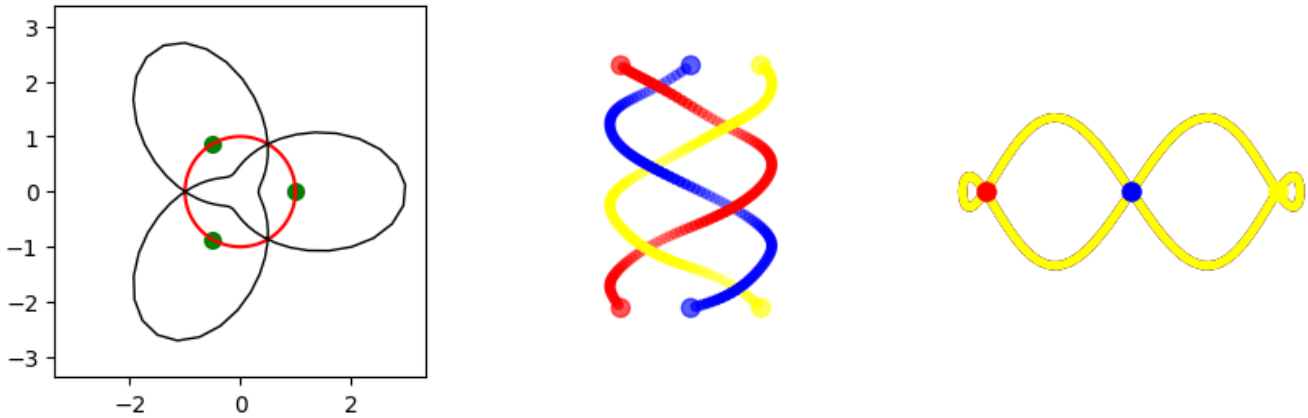
>>> **tb.plot_ThreeBraid('01', 'all', connectSZ=True, T=(0,4))**



>>> **tb.plot_ThreeBraid('01', 'all', connectSZ=True, T=(0,5))**



```
>>> tb.plot_ThreeBraid('01', 'all', connectSZ=True, T=(0,6))
```



６倍して shape sphere 上の閉路になりました。途中３倍したところでも、始点と終点が一致してはいますが、道の向かう向きが反対向きなので、さらに３倍をつないだ６倍長のものの方が適切でしょう。

AB 型表現形式など、さまざまな形で与えられた 3-braid に対しても **tb.plot_ThreeBraid** 関数で、3-braid の３次元像（**'braid'**）、３動点の軌跡（**'orbit**）、shape sphere 上の道（**'pathSS', 'path'**）を描くことができます。

[図のサイズ、配色など]　　図のサイズ、配色などは、**plot()** メソッドの引数に指定することで変更することが可能です。ほとんどは **matplotlib** モジュールのオプションと共通にしていますが、一つ一つ覚えておいて、描画の度にいちいち書き込むのは結構面倒です。**tb.setOption()** 関数を用意しました。この関数を実行すると、幾つか質問されます。変更したい部分について入力することで、以後、その設定で描がかれるようになります。変更しなくてもいい部分については、何も入力せず、[return]/[enter] キーを押し、次に質問に移ってください。また、**plot()** メソッドで指定したい描画用のキーワード引数とその値を、**tb.setOptions()** 関数の引数にすることで、その値のみを直接変更することもできます。冗長になるので、**plot()** メソッドで指定可能な描画用キーワード引数については省略します。

[図の保存方法]　　視覚化手続きで表示された図を簡単に保存することができます。図の右の **Save to** の右の欄にファイル名を書き、その右の **Save** ボタンを押すだけで、左に表示されているグラフを png 形式のファイルで保存することができます。他にも eps, jpg (jpeg), pdf, pgf, ps, raw, rgba, svg, svgz, tif (tiff) 形式に対応しています。拡張子に指定することで、それぞれの形式で保存できます。

In [ ]: