pgfid16@offset pgfid18@offset pgfid20@offset

THE NON-PLANARITY OF SL$(2, \mathbb{Z})$-ORBITS OF ORIGAMIS

DAVID JIANG

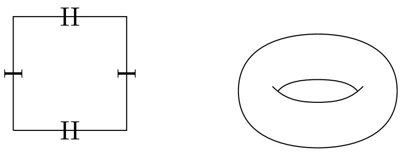
Abstract. An origami is a mathematical construct to help understand the fundamental underpinnings of topology. In this paper we explore a new method for applying transformations to new origamis and look at their orbit graphs to see non-planarity. This will help us gain new further evidence for things such as McMullen conjectures.

Contents

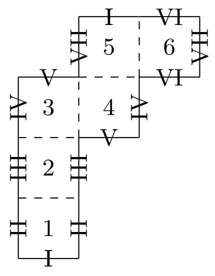
1. Introduction

1.1. **Origamis.** We can define connected unit squares in $\mathbb{R}^2$ with a ruleset defined as to where edges or squares are connected to each other.

Commonly shown in topology classes, a genus-1 torus can be represented as an origami as follows:



Where the sides labelled with I are connected with each other and II is connected with II. This can be shown with a piece of paper yourself or through visualization. However, these origamis will grow rapidly where it no longer is feasible to do it with paper yourself or through visualization such as:

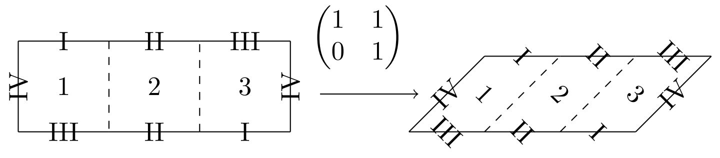


This is defined as a 6 squared origami and will be one of the main classes of origamis that we will look at.

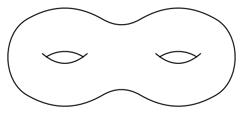
1.2. **Transformations.** We can act on these origamis by the generators of $\mathrm{SL}(2,\mathbb{Z})$

$$S = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}, T = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

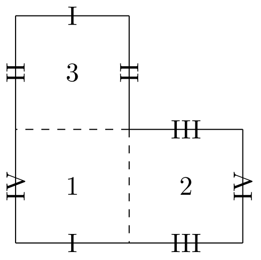
The group action of the generators on our origamis is the same as the linear transformation on $\mathbb{R}^2$ that the matrices normally have. Where T is a horizontal shear and S is a vertical shear. For example, we will look at



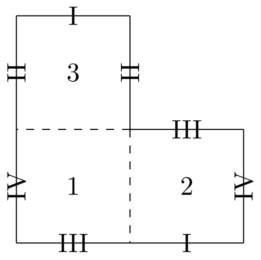
We notice that nothing has changed about this origami because we can connect each side with each side. Both of these origamis will result in a genus-2 object such as:



We note that there exist other 3 square origamis that can be folded into this object as such:



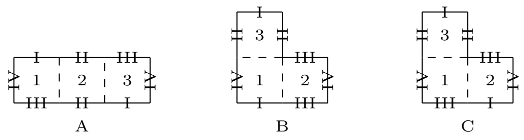
B



C

These origamis behave slightly different when acting with our generators on these. Running through the transformations through T and S and then seeing what origami they turn into (they aren't all identity transformations) gives us the following orbit graph:

$$\begin{tikzcd}
A \arrow[loop left, "T"] \arrow[r, bend left, dash, "S"] \arrow[r, bend right, dash, "S"'] & B \arrow[r, bend left, dash, "T"] \arrow[r, bend right, dash, "T"'] & C \arrow[loop right, "S"]
\end{tikzcd}$$



1.3. **Permutation Groups.** All of our origamis are uniquely defined by their permutation group representation. Another way to think about the origami is instead of folding the papers, each edge labeled corresponds to a door to the other edge. We then think of the permutation where we shift everything either horizontally or vertically. This is what actually defines our orbits. For example, B can be described has $\langle h, v\rangle = \langle (1,2),(1,3)\rangle$ since when shifting everything horizontally, we notice that $1 \to 2$ while $3 \to 3$, which gives us the transposition $(1,2)$ for a horizontal movement. Similarly, vertical gives us $(1,3)$. We notice that origami A corresponds to $\langle (1,2,3),(1,3)\rangle$ and C is $\langle (1,2),(1,3,2)\rangle$. So we notice that all the nodes on our orbit graph correspond to a unique permutation representation of $S_3 \oplus S_3$.

1.4. **Graph Theory.** Although we've already mentioned graphs from orbit graphs, we will give a more formal introduction. A graph is defined as a set of nodes and edges where edges are connected to various nodes. For example, the graph that we previously had is described as:

$$\begin{tikzcd}
A \arrow[loop left, "T"] \arrow[r, bend left, dash, "S"] \arrow[r, bend right, dash, "S"'] & B \arrow[r, bend left, dash, "T"] \arrow[r, bend right, dash, "T"'] & C \arrow[loop right, "S"]
\end{tikzcd}$$

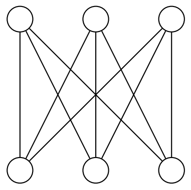
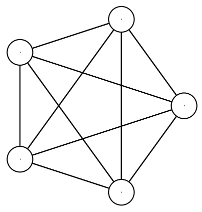
Where we have the nodes A, B, and C and then the edges are described by the arrows. We now have the machinery required to describe the problem that we want to look at. When generalizing an $n-$square origami, is it possible to create and prove a general permutation representation that will result in the entire orbit graph to be non-planar.

Planar graphs are graphs that can be drawn on the plane in such a way that no edges intersect one another.

Roughly speaking, we are interested in showing that a certain infinite family of graphs is eventually non-planar. This will give indirect evidence that the family could form a family of expander graphs (useful objects in many areas of computer science and mathematics).

The outline of the way we will prove non-planarity is invoking **Kuratowski-Wagner Theorem**.

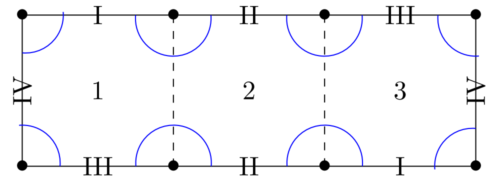
Theorem 1.1. *A graph is planar if and only if it does not contain $K_{3,3}$ (left) or K_5 (right) as a graph minor (obtained by deleting edges, vertices and by contracting edges).*

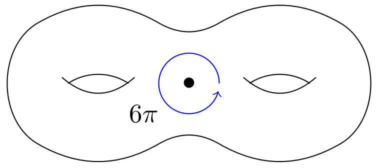



So if we can find a $K_{3,3}$ or K_5 in our orbit graphs, then we can invoke our theorem and therefore it would be non-planar. This has already been done for certain graphs by Jeffreys and Matheus [JM], now we will try to replicate the process for graphs with stratum $\mathcal{H}(4)$ instead of $\mathcal{H}(2)$ as described in their paper.

1.5. **Strata.** The vertices of the squares used to make an origami can have more than 2π angle around them. We call such points cone points. We say that an origami lies in the **stratum** $\mathcal{H}(k_1, k_2, ..., k_n)$ if it has n cone points, and each has cone angle $2\pi + k_i \cdot 2\pi$. The Gauss-Bonnet (or Riemann-Roch) Theorem allows us to determine the genus g through the formula $\sum_{i=1}^{n} k_i = 2g - 2$.

Thus, for example, any surface in $\mathcal{H}(2)$ should have genus 2, and it has only one cone point with angle 6π, as follows:





Our focus for this project will be origamis lying in the stratum $\mathcal{H}(4)$. So, since $4 = 2 \cdot 3 - 2$, we have origamis of genus 3 with a single cone point.

2. Implementation

Our goal of the project as mentioned is to observe non-planarity in orbit graphs. What we will do is use various packages in Python that allow us to generate orbit graphs for higher square-tiled surfaces and then take 6 different nodes and see if they are in a $K_{3,3}$. This will give us the information we need to know when a certain $n-$tiled surface is non-planar. From there, we will try to generalize the permutation pattern that we have and extrapolate for larger n.

2.1. **Sage Math and Surface Dynamics.**

We worked primarily through Python with the packages Sage [SM] and Surface Dynamics. Sage is an open source library that is built ontop of NumPy, SciPy, matplotlib, and many other academic Python packages. It functions as a computer algebra system (CAS) which allows plenty of tools to work with when looking at algebraic, combinatoric, or graph theoretic questions.

Surface Dynamics [SD] is another python package built off the works of Sage that works on specific problems of math. Surface dynamics allows us to work with square-tiled surfaces, origamis, graphs and strata. It allows us to have access to databases of origamis and allows quick computation of orbit graphs and permutation representations of various origamis. These two packages provide us all that we need in order to create programs that will tell us if we have non-planar orbit graphs.

2.2. **Our Code.**

```python
from surface_dynamics import *
import itertools
D = OrigamiDatabase()
n = 6
q = D.query(nb_squares = n, stratum = AbelianStratum(4))
for o in q:
    if o.is_primitive() and o.orientation_data() == [] and o.monodromy().order() == factorial(n)/2:
        O = o
        C = O.teichmueller_curve()
        G = C.orbit_graph(r_edges = True, l_edges = True, s2_edges = False, s3_edges = False)
        G.show()
        G = G.to_simple()
        G = G.to_undirected()
        print(G.is_planar())
        for trip_1 in itertools.combinations(G,3):
            G_hat = [x for x in G if not x in trip_1]
            for trip_2 in itertools.combinations(G,3):
                disjoint_check = True
                for o in trip_1:
                    if o in trip_2:
                        disjoint_check = False
                        break
                if not disjoint_check:
                    pairs = [list(z) for z in itertools.product(trip_1, trip_2)]
                    try:
                        G.disjoint_routed_paths(pairs)
                    except EmptySetError:
                        continue
                else:
                    for i in range(1,4):
                        print(f"O_{i} = ({trip_1[i-1].r()}, {trip_1[i-1].u()})")
                    for i in range(1,4):
                        print(f"O_{i+3} = ({trip_2[i-1].r()}, {trip_2[i-1].u()})")
                    print()
```

I will go through line by line explaining what exactly this code accomplishes.

As previously mentioned, surface dynamics gives us a database of origamis. We will use this database and look at the origamis for $n = 6$ as a starting point. We want to clarify that the origami has exactly 6 sqaures and that it has a stratum of $\mathcal{H}(4)$. With these parameters, q now provides us with all of our origamis that we would want to look at. We are primarily interested in primitive origamis, which are origamis that are transitive with only trivial partitions. So the first if statement allows us to prune all of the origamis that we aren't interested in looking at.

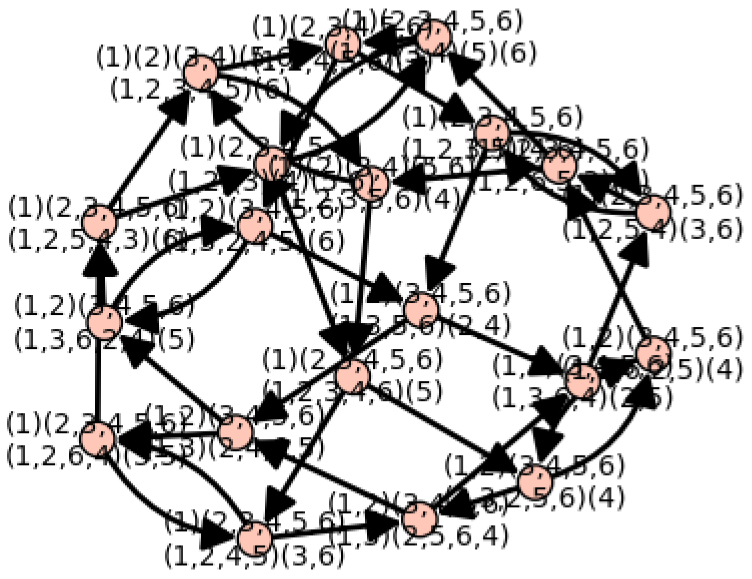
Then the first 4 lines after the if statement generate the orbit graph that we are curious in. The parameters are just extra things that allow for easier analysis. We next convert this graph into a simple undirected graph so our itertools can work some graph theory magic on it.

Afterwards we starting to use itertools to select 3 nodes on our orbit graph. These will denote the first set of nodes in our potential $K_{3,3}$. Then we select another 3 nodes to be our second set for our $K_{3,3}$. We want to ensure that these are disjoint, so as the name would imply, disjoint check accomplishes this for us.

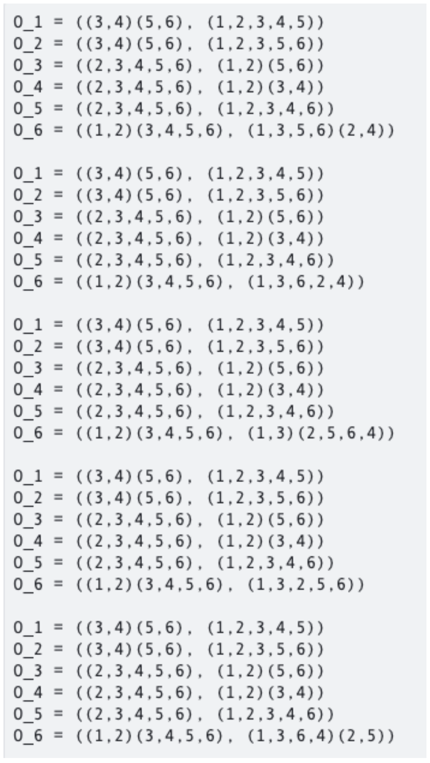
Now we will pair up all possible pairs from set 1 and set 2 through itertools product. This is just taking the cartesian product between the two sets. We will try to find a disjoint path from one node in the firts set to the other nodes in the second set. These will have to be disjoint in order for the minor requirements to hold. If there is an error, then we will go onto the next set of nodes. Otherwise, we will print all of the possible results that result in a non-planar graph.

3. Current Results

3.1. **Data.** From running the code shown above, we get that there are thousands of different nodes that give us a valid non-planar $K_{3,3}$. For perspective, this is the orbit graph of 6 tiles:

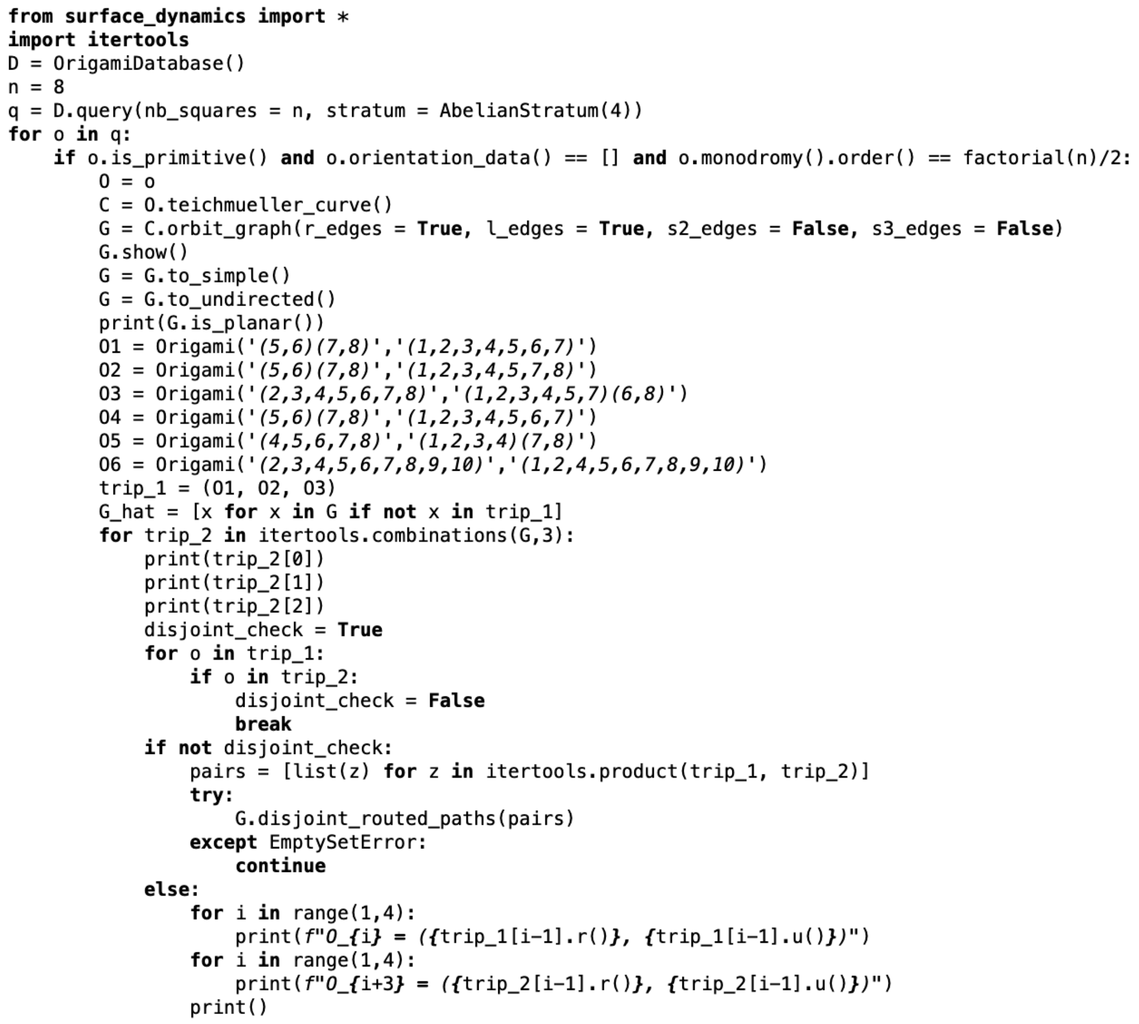


As an example of the various nodes that work for this graph, we have the following few choices:



```
O_1 = ((3,4)(5,6), (1,2,3,4,5))
O_2 = ((3,4)(5,6), (1,2,3,5,6))
O_3 = ((2,3,4,5,6), (1,2)(5,6))
O_4 = ((2,3,4,5,6), (1,2)(3,4))
O_5 = ((2,3,4,5,6), (1,2,3,4,6))
O_6 = ((1,2)(3,4,5,6), (1,3,5,6)(2,4))

O_1 = ((3,4)(5,6), (1,2,3,4,5))
O_2 = ((3,4)(5,6), (1,2,3,5,6))
O_3 = ((2,3,4,5,6), (1,2)(5,6))
O_4 = ((2,3,4,5,6), (1,2)(3,4))
O_5 = ((2,3,4,5,6), (1,2,3,4,6))
O_6 = ((1,2)(3,4,5,6), (1,3,6,2,4))

O_1 = ((3,4)(5,6), (1,2,3,4,5))
O_2 = ((3,4)(5,6), (1,2,3,5,6))
O_3 = ((2,3,4,5,6), (1,2)(5,6))
O_4 = ((2,3,4,5,6), (1,2)(3,4))
O_5 = ((2,3,4,5,6), (1,2,3,4,6))
O_6 = ((1,2)(3,4,5,6), (1,3)(2,5,6,4))

O_1 = ((3,4)(5,6), (1,2,3,4,5))
O_2 = ((3,4)(5,6), (1,2,3,5,6))
O_3 = ((2,3,4,5,6), (1,2)(5,6))
O_4 = ((2,3,4,5,6), (1,2)(3,4))
O_5 = ((2,3,4,5,6), (1,2,3,4,6))
O_6 = ((1,2)(3,4,5,6), (1,3,2,5,6))

O_1 = ((3,4)(5,6), (1,2,3,4,5))
O_2 = ((3,4)(5,6), (1,2,3,5,6))
O_3 = ((2,3,4,5,6), (1,2)(5,6))
O_4 = ((2,3,4,5,6), (1,2)(3,4))
O_5 = ((2,3,4,5,6), (1,2,3,4,6))
O_6 = ((1,2)(3,4,5,6), (1,3,6,4)(2,5))
```

3.2. **Results.** We want to be able to generalize these data points for $K_{3,3}$. So looking at some of the points in the previous section, we notice that the first point can be generalized as two transpositions between $(n-1, n)$ and either $(n/2, n/2+1)$ or $(n-3, n-2)$. We can do something similar for several other nodes.



```
from surface_dynamics import *
import itertools
D = OrigamiDatabase()
n = 8
q = D.query(nb_squares = n, stratum = AbelianStratum(4))
for o in q:
    if o.is_primitive() and o.orientation_data() == [] and o.monodromy().order() == factorial(n)/2:
        O = o
        C = O.teichmueller_curve()
        G = C.orbit_graph(r_edges = True, l_edges = True, s2_edges = False, s3_edges = False)
        G.show()
        G = G.to_simple()
        G = G.to_undirected()
        print(G.is_planar())
        O1 = Origami('(5,6)(7,8)','(1,2,3,4,5,6,7)')
        O2 = Origami('(5,6)(7,8)','(1,2,3,4,5,7,8)')
        O3 = Origami('(2,3,4,5,6,7,8)','(1,2,3,4,5,7)(6,8)')
        O4 = Origami('(5,6)(7,8)','(1,2,3,4,5,6,7)')
        O5 = Origami('(4,5,6,7,8)','(1,2,3,4)(7,8)')
        O6 = Origami('(2,3,4,5,6,7,8,9,10)','(1,2,4,5,6,7,8,9,10)')
        trip_1 = (O1, O2, O3)
        G_hat = [x for x in G if not x in trip_1]
        for trip_2 in itertools.combinations(G,3):
            print(trip_2[0])
            print(trip_2[1])
            print(trip_2[2])
            disjoint_check = True
            for o in trip_1:
                if o in trip_2:
                    disjoint_check = False
                    break
            if not disjoint_check:
                pairs = [list(z) for z in itertools.product(trip_1, trip_2)]
                try:
                    G.disjoint_routed_paths(pairs)
                except EmptySetError:
                    continue
            else:
                for i in range(1,4):
                    print(f"O_{i} = ({trip_1[i-1].r()}, {trip_1[i-1].u()})")
                for i in range(1,4):
                    print(f"O_{i+3} = ({trip_2[i-1].r()}, {trip_2[i-1].u()})")
                print()
```

We now use the following code depicted above to test those 6 points that we have attempted to generalize for an $n = 8$ orbit graph. Some common issues that get run into is the startum not being correct. Sometimes you find that the origami you tried to generalize to is actually of $\mathcal{H}(2)$ or $\mathcal{H}(2,2)$. So even though these generalized nodes might be non-planar for our $K_{3,3}$ they are still not correct because they don't apply directly to the problem that we are trying to solv.

Another approach we can take is find several points that seem to appear in all of our various $K_{3,3}$ for $n = 6$ and then try to generalize those simply and then running some sort of iteration between the others to find potential $K_{3,3}$ in $n = 8$ instead. This circumvents the issue of being worried about the strata of the origamis because we can prefilter all of the strata. We have not attempted this approach enough yet but we will attempt to do so next semester or over winter break.

There are various other things we can try. It makes sense to actually look at the origamis that show up often in $n = 6$. To do so we can display them using surface dynamics. Looking at these we can instead try to generalize based on the actual square tilings instead of the square tiles themselves.

3.3. **Open Questions.**

We have still not found a way to generalize for $n = 8$ and other even amounts. We additionally want to do something similar for the odd numbers. We imagine there would be something similar, but it isn't directly generalized from evens to odd since a lot of ways of generalization are built on the parity of the numbers that we are working with.

Additionally, once we find out what generalization we have then we will also need to prove that they will always give you a non-planar graph. The issue is that originally finding a way to prove that the orbit graphs were non-planar without finding a generalized set of 6 nodes that satisfy our $K_{3,3}$. This will make it signficantly easier to prove.

Another thing to think about is the orbit graphs of the other generators for $\mathrm{SL}(2, \mathbb{Z})$. They will generate various other orbit graphs, however, it's expected that they will give more gross results.

[1] V. et al. Delecroix. surface dynamics - SageMath package, Version 0.4.7. July 2021. DOI: 10.5281/zenodo.3237923. URL: https://doi.org/10.5281/zenodo.3237923.

[2] Luke Jeffreys and Carlos Matheus. “Non-planarity of SL(2, Z)-orbits of origamis in H(2)”. In: Bulletin of the London Mathematical Society 55 (May 2023). DOI: 10.1112/blms.12849.

[3] Carlos Matheus, Martin Moeller, and Jean-Christophe Yoccoz. “A criterion for the simplicity of the Lyapunov spectrum of square-tiled surfaces”. In: Inventiones mathematicae 202 (May 2013). DOI: 10.1007/s00222-014-0565-5.

[4] The Sage Developers. SageMath, the Sage Mathematics Software System (Version 10.2). https://www.sagemath.org. 2023.