Operations on generic plane curves

Bjørnar Gullikstad Hem
Supervised by Markus Szymik

Norwegian University of Science and Technology, Trondheim, Norway

2020

1 Introduction

In this project, I explore generic plane curves, operations to transform different generic plane curves into each other, and the equivalence classes these operations create. I consider three types of modifications, *move 1*, *move 2*, and *move 3*, that correspond to the three types of Reidemeister moves for knot diagrams. I show that move 1 and move 2 can be used to construct a generic plane curve from the curve's unique *1–2–reduced* curve, and that there exists several 1–2–reduced curves. I further show that all generic plane curves can be constructed from the *trivial curve* with the three moves. I also implement a result from [Ghy17, p. 232-236] to count the number of generic plane curves with n crossings.

A plane curve is a map $\mathbb{R} \to \mathbb{R}^2, t \mapsto (x(t), y(t))$ that is continously differentiable, with non-zero derivative, and such that $(x(t), y(t)) = (t, 0)$ for large $|t|$. A plane curve is *generic* if every point (x, y) is passed at most twice, and if so, the derivatives there are linearly independent, so that we have a *crossing*.

Two generic plane curves are equivalent if they can be continuously transformed into each other through generic plane curves.

Let the *trivial curve* be defined as the generic plane curve with no crossings.

Figure 1: Trivial curve

Let *simple curve 1* and *simple curve 2* be defined as the two generic plane curves with one crossing, as shown in figures 2 and 3.

When defining the following moves, I will also consider curves with a beginning and an end, that would be generic plane curves if the endpoints were "stretched out" to $-\infty$ and $+\infty$.

Let *move 1* consist of transforming a trivial curve that does not intersect any other curves into either simple curve 1 or simple curve 2 such that the resulting curve does not intersect any other curves, while keeping the endpoints fixed.

Figure 2: Simple curve 1

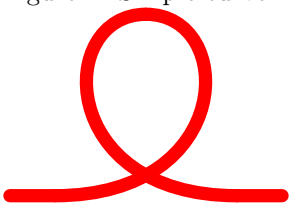


Figure 3: Simple curve 2

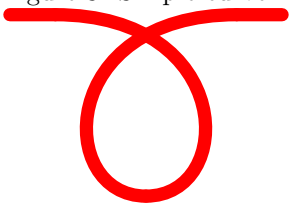
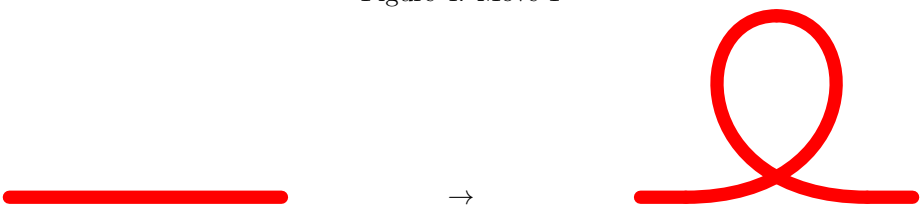
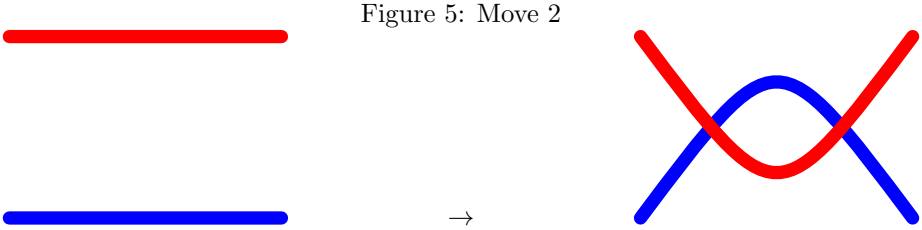


Figure 4: Move 1



Let *move 2* consist of transforming two trivial curves that do not intersect any other curves into two trivial curves that intersect each other twice and do not intersect any other curves, while keeping the endpoints fixed.

Figure 5: Move 2



One can use move 1 and move 2 to create more complicated curves, starting with the trivial curve, as illustrated in the figures 6, 7 and 8. Clearly a curve created from applying move 1 or move 2 to a generic plane curve is a generic plane curve. However, there are generic plane curves that can not be created from the trivial curve using only these moves (repeatedly).

Figure 6: An example of using move 1 twice to create a generic plane curve with two crossings.

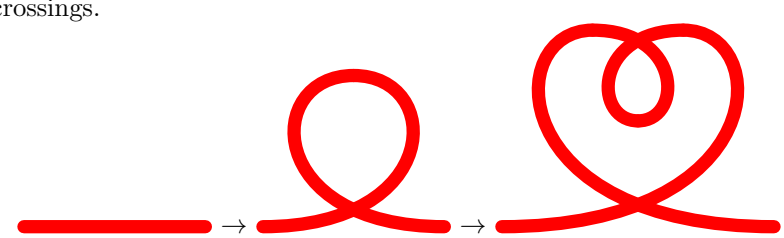


Figure 7: Another example of using move 1 twice to create a generic plane curve with two crossings.

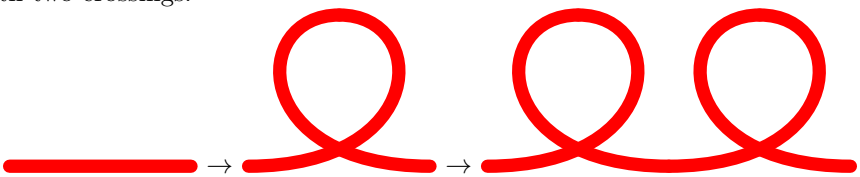
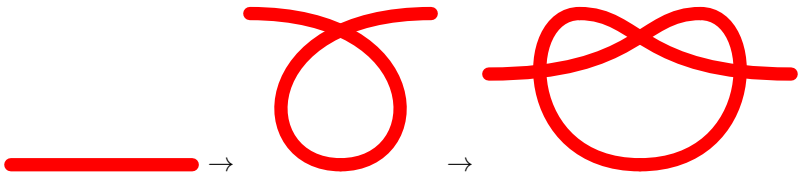


Figure 8: An example of using move 1 followed by move 2 to create a generic plane curve.



Let the inverses of move 1 and move 2 be *reversed move 1* and *reversed move 2*.

Let two generic plane curves be *1–2–related* if there is a sequence of moves of type 1 or 2 (or their inverses) that relates them.

Theorem 1. *There are generic plane curves that are not 1–2–related to the trivial curve.*

Let a generic plane curve be *1–2–reduced* if neither reversed move 1 nor reversed move 2 can be applied to it.

Theorem 2. *Every generic plane curve is 1–2–related to a unique 1–2–reduced generic plane curve.*

Let *move 3* consist of transforming three trivial curves such that the first two intersect the third once each and intersect each other exactly once and on one side of the third and such that the trivial curves intersect no other curves, into three trivial curves such that the first two intersect the third once each and intersect each other exactly once and on the other side of the third and such

that the trivial curves intersect no other curves, all while keeping the endpoints fixed, as shown in figure 9.

Figure 9: Move 3

Theorem 3. *All generic plane curves can be created from the trivial curve using only moves of type 1, 2 or 3.*

2 Proof of theorem 1

Proof. Suppose a generic plane curve is created by repeatedly applying one of move 1 and move 2 to the trivial curve. If the last move was move 1, then the curve must enclose an area that has exactly one crossing on its boundary, as such an area was created with the last move. Similarly, if the last move was move 2, then the curve must enclose an area that has exactly two crossings on its boundary. Therefore, a generic plane curve that contains no areas with either one or two crossings on its boundary cannot be constructed from the trivial curve using only move 1 and move 2. Figure 10 shows an example of a generic plane curve that contains no areas with either one or two crossings on its boundary, which completes the proof.

□

3 Proof of theorem 2

Let us now define *signed Gauss words*, as described in [Ghy17, p. 232-236]. For a generic plane curve, assign to each crossing a symbol, $a_i, i \in \mathbb{N}$. If one moves along the curve from left to right, and writes down the symbol a_i for the crossing each time a crossing is passed, one gets a *Gauss word* for the curve. The *Gauss word* is of length $2n$, where n is the number of crossings, and each symbol a_i appears twice. Each crossing can be thought of as two directed curve segments crossing each other, and each occurence of a symbol a_i corresponds to one of these segments being traversed. A *signed Gauss word* is a Gauss word where each occurence of a symbol is assigned either $+$ or $-$ depending on the direction of the segment being crossed (see [Ghy17, p. 233] for an example and an illustration). Every generic plane curve can be described with a signed Gauss word and every signed Gauss word uniquely defines a generic plane curve.

Move 1 can be written in the language of signed Gauss words as adding either $a_i^+ a_i^-$ or $a_i^- a_i^+$ somewhere in the word (and reversed move 1 as the inverse of

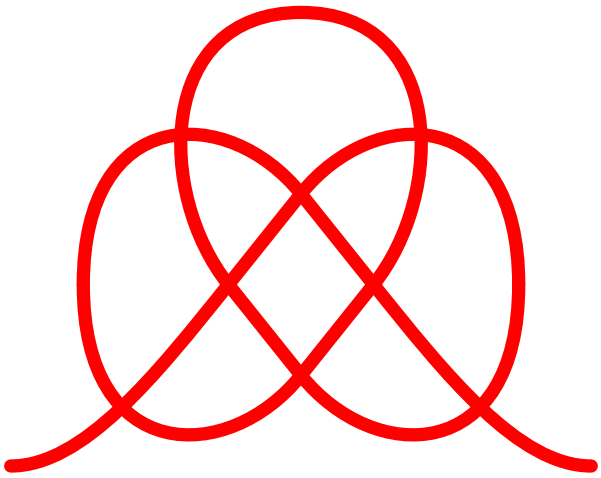


Figure 10: A generic plane curve that cannot be constructed from the trivial curve using only move 1 and move 2.

this). Move 2 can be written in the language of signed Gauss words as adding both $a_i^+ a_j^-$ and $a_j^+ a_i^-$, both $a_i^- a_j^+$ and $a_i^+ a_j^-$, both $a_i^+ a_j^-$ and $a_i^- a_j^+$ or both $a_i^- a_j^+$ and $a_j^- a_i^+$ somewhere in the word (and reversed move 2 can be written as the inverse of this).

Proof. Assume for contradiction that a 1–2–reduced generic plane curve is 1–2–related to a different 1–2–reduced generic plane curve. Then the signed Gauss words of these curves must be different. Thus, there must be a sequence of transformations on the first Gauss word, consisting of only move 1, move 2 and their inverses, that adds a new letter a_n to the word, which cannot be removed with reversed move 1 or reversed move 2.

For this to happen, at some point in the sequence a transformation must be made that makes a letter a_k go from being removable with a sequence of reversed move 1 and reversed move 2 to not being so. The only one of move 1, move 2, reversed move 1 and reversed move 2 that can do this is reversed move 2. This can only happen if both pairs of letters, $a_i^\pm a_j^\pm$ or $a_j^\pm a_i^\pm$, is adjacent to a letter with the same index, $a_k^\pm$, at the same index, like so: $a_i^\pm a_j^\pm a_k^\pm$ or $a_k^\pm a_j^\pm a_i^\pm$. However, in this case, removing $a_j^\pm$ and $a_i\pm$ and removing $a_j^\pm$ and $a_k^\pm$ would result in the same Gauss curves, just with the indexes i and k replaced.

□

4 Proof of theorem 3

A generic plane curve can be viewed as a planar embedding of a graph, where the vertices are the crossings and the edges are the arcs connecting the crossings.

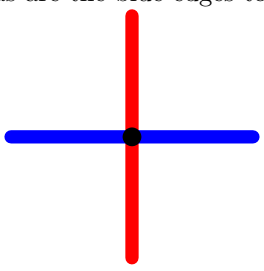
The arc going to $(-\infty, 0)$ can be considered connected to the arc going to $(\infty, 0)$, so that they form a single edge connecting two vertices (crossings). Then each node (crossing) in the plane graph has exactly four edges. Let the *plane graph* of a generic plane curve be this planar embedding of a graph representing the generic plane curve.

To see that theorem 3 is true, consider knot diagrams, as described in [AB26] and [Rei27]. These can be viewed as plane graphs of generic plane curves with the added structure that for each crossing, two opposite edges are "above" and two are "below". Notice that the three Reidemeister moves (deformation type II, III and IV in [AB26, p. 565-567]) for knot diagrams transform the diagram in the same way as move 1, move 2 and move 3 transform the corresponding generic plane curve. Thus, if a knot diagram corresponding to a generic plane curve can be transformed to a knot diagram with zero crossings with the three Reidemeister moves, then the generic plane curve can be transformed to the trivial curve with move 1, move 2 and move 3. A knot diagram that satisfied this can be chosen for any generic plane curve, as follows: Choose the knot diagram such that at each crossing, the edges that are passed second when moving along the curve from left to right are "above". Then the given knot is of the same type as the trivial knot. We know from [AB26, p. 565-567] that knot diagrams of the same type can be transformed into each other with the Reidemeister moves, so the knot diagram can be transformed to a knot diagram with zero crossings using the Reidemeister moves.

A proof that does not require the theory of Reidemeister moves follows on page 12.

I will here define what I mean when I write that two edges are *opposite* to each other with respect to a vertex. Picking an arbitrary edge from the four edges indicent to a vertex, and moving counter-clockwise (or clockwise) around the vertex, the first and third edge are *opposite* to each other with respect to that vertex, and so are the second and fourth edge.

Figure 11: Four edges (red and blue) incident to a vertex (black). The red edges are opposite to each other, as are the blue edges to each other.



In the rest of this section, I wille use the following terminology for graphs: A *trail* can have repeated vertices, but not repeated edges. A *circuit* is a trail that begins and ends on the same vertex. A circuit is *simple* if the only repeated vertex is the first/last vertex, and that vertex is only visited at the beginning

and end.

Consider the trails in said plane graph. Let a *corner* in a trail be a vertex in the trail where the edge traversed before and after it are not opposite to each other with respect to the vertex. Let the *degree* of a trail be the number of corners in the trail. For a circuit, the starting vertex is considered a corner if the first and last edge in the circuit is not opposite to each other with respect to that vertex. Notice that in a plane graph of a generic plane curve, the only circuit of degree 0 is a circuit consisting of all the edges in the graph.

Let the *interior* of a circuit be the closed region bounded by the circuit in the plane graph.

Let the *chords* of a circuit be the trails of degree 0 that begin and end on vertices in the circuit, and whose edges lie entirely in the interior of the circuit.

Let a circuit L be *n–minimal* if there are no circuits of degree n or less in the interior of L except perhaps L itself. It follows from the definition that if $k < n$, a n–minimal circuit is k–minimal.

Note that in a 1–minimal circuit, no chord may intersect itself. Furthermore, in a 2–minimal circuit, no two chords may intersect each other more than once.

Figure 12: Three generic plane curves. In blue: From left to right: a simple circuit of degree 1, 2 and 3, respectively. In green: a chord in the blue circuit.

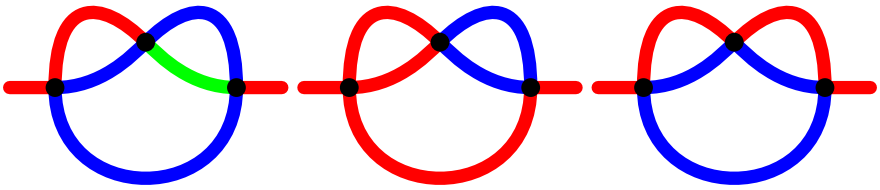
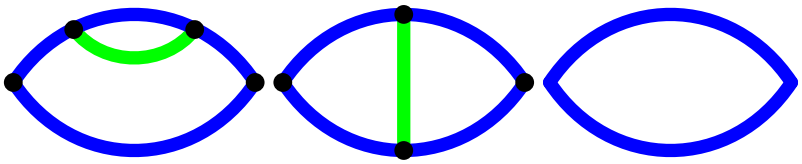


Figure 13: From left to right: a 1–minimal simple circuit, a 2–minimal simple cicuit and a simple circuit that has no chords (all of degree 2).



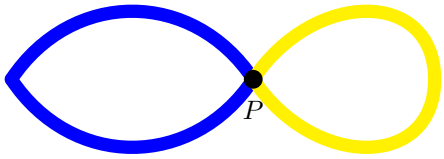
Lemma 1. *Let L be a circuit of degree 1. Then there is a simple circuit of degree 1 in L.*

Proof. If L is simple, L is a simple circuit of degree 1 in L. Assume L is not simple. Let P be the first vertex that is passed twice when moving along L from the corner in L. Then the trail in L connecting P to itself is simple, and it is of

degree 1 because there can be no other corners in it than P (then L would not be of degree 1) and it can not be of degree 0.

□

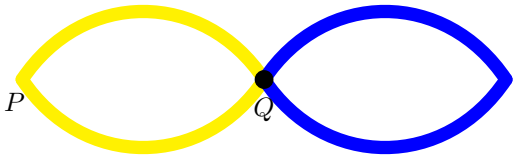
Figure 14: Illustration for lemma 1. A simple circuit of degree 1 (yellow) in a non-simple circuit of degree 1 (blue and yellow).



Lemma 2. *Let L be a circuit of degree 1 or 2. Then there is a simple circuit of degree 1 or 2 in L.*

Proof. If L is simple, L is a simple circuit of degree 1 or 2 in L. Assume L is not simple. Let P be a corner in L. Let Q be the first vertex that is passed twice when moving along L from P. Let M be the circuit in L connecting Q to itself. Then M is simple, and it cannot be of degree higher than 2, as all corners in M except for perhaps Q are also corners in L, and P is not in M. □

Figure 15: Illustration for lemma 2. A simple circuit of degree 2 (yellow) in a non-simple circuit of degree 2 (blue and yellow).



Lemma 3. *Let L be a simple circuit of degree 1 or 2. Then there is a 2–minimal simple circuit of degree 1 or 2 in the interior of L.*

Proof. Assume L is not 2–minimal. Let L' be a circuit of degree 1 or 2 in the interior of L that is not L itself. There are two cases to consider (illustrated in figure 17):

1. At least one edge in L is not in L'
2. All edges in L are also in L'

If L' falls into the second case, we in stead pick a new candidate for L' that falls into the first case. We can do this, as if all edges in L are also in L', and L' is not equal to L, then there must be a repeat of a vertex in L' other than the repeat of the first/last vertex (to see this, consider an edge in L' not in L that succeeds an edge in L, and notice that the fist vertex of this edge must be repeated in L'). This ensures that any circuit that is in L' is different from L.

By lemma 2 there is a simple circuit of degree 1 or 2 in L', not necessarily different from L', call it L''. L'' is also in the interior of L. Let S and S'' be the number of regions that the edges and chords of L and L'' divide the plane into, respectively. Because L'' is in the interior of L, and not equal to L', S'' is smaller than S. By the principle of infinite descent, there is a 2–minimal simple circuit of degree 1 or 2 in L. □

Figure 16: Illustration for lemma 3. An example of the algorithm described to find a 2–minimal simple circuit of degree 2 in the interior of a simple circuit of degree 2.

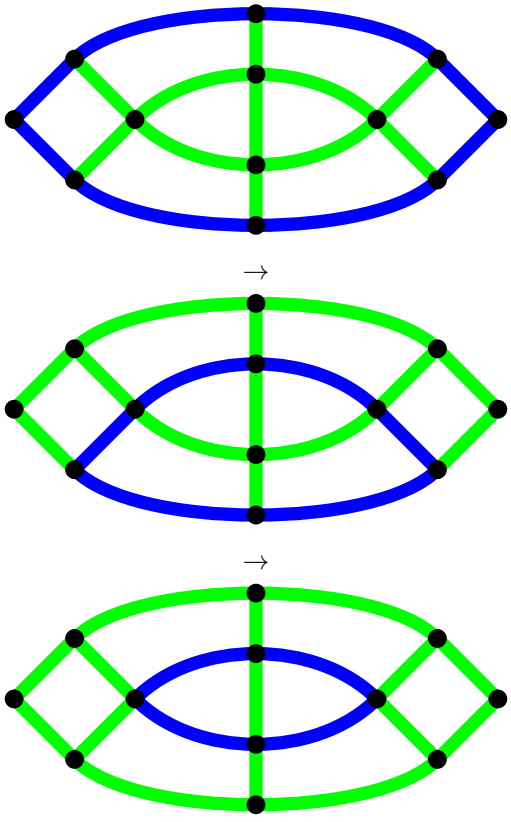
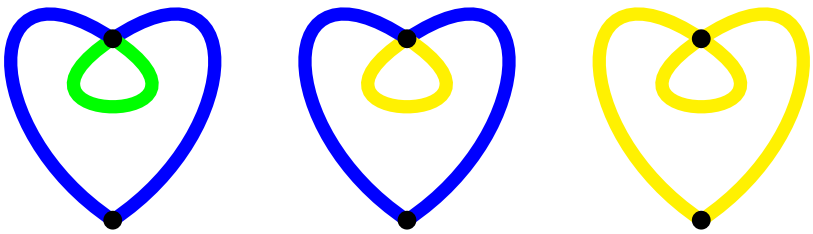


Figure 17: Illustration for lemma 3. Left: in blue: L, a simple circuit of degree 2. Middle: Yellow: L' in case 1 from the proof. Right: Yellow: L' in case 2 from the proof.



Lemma 4. *In a plane graph of a generic plane curve with one or more crossings, there is a 2–minimal simple circuit of degree 1 or 2.*

Proof. Let C be a plane graph of a generic plane curve with at least one crossing, and let P be a crossing on C. The arc on C connecting P to itself is a circuit of degree 1. By lemmas 1 and 3 there is a 2–minimal simple circuit of degree 1 or 2 in C. □

Lemma 5. *Let L be a 2–minimal simple circuit of degree 3. Assume that there is an edge in L whose endpoints are corners in L, and let a be the degree 0 trail connecting a different pair of corners corners in L. Then there is a simple circuit of degree 3 that has no chords and lies in the interior of L that shares an edge with a.*

Proof. Assume L contains at least one chord. Let A, B and C be the corners of L such that B and C are the endpoints of a and there is an edge in L between A and B (that is, no chords intersect AB). Furthermore, let b be the trail in L connecting A and C with degree 0 (i. e. that does not contain B), and let P be the vertex in b adjacent to C. Let l be the chord with P as an endpoint, and let Q be the other endpoint of l. Let L' be the circuit consisting of edges in L and l that contains C. Then C, P and Q are the corners of L', and L' is of degree 3. Because L is 2–minimal, l does not intersect itself, so L' is simple. Because L' is in the interior of L, it is 2–minimal.

Then L' is a 2–minimal simple circuit of degree 3 in the interior of L that shares an edge with a and has a different edge whose endpoints are corners in L'. Furthermore, the interior of L' has fewer crossings than the interior of L because it does not include A and B and because L' is inside L. One can thus continue this process until one gets a simple circuit of degree 3 in the interior of L that contains no chords and shares an edge with a.

□

Figure 18: Illustration for lemma 5.

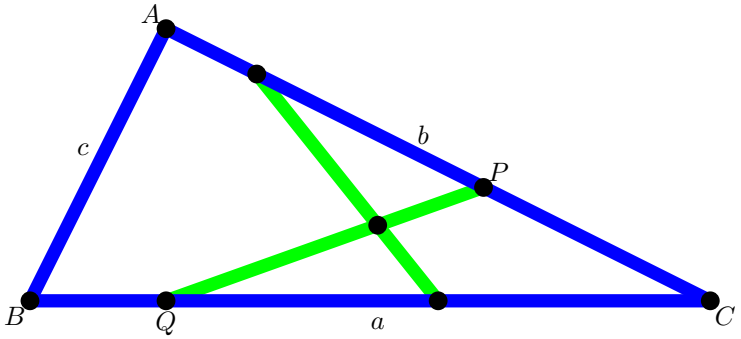
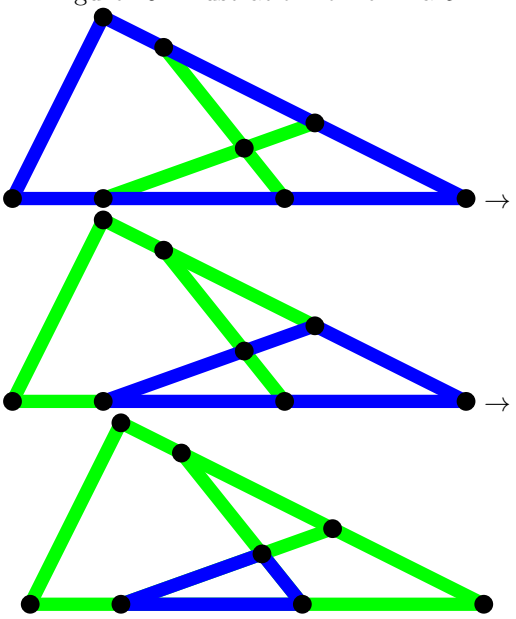


Figure 19: Illustration for lemma 5.



Lemma 6. *Let L be a 2–minimal simple circuit of degree 1 or 2. If the open region bounded by L contains a vertex, the interior of L contains a simple circuit of degree 3 that contains no chords and shares an edge with L.*

Proof. Let l be a chord of L that has at least one vertex different from its endpoints and let E and F be its endpoints. Let A and B be the crossings on l, not necessarily different, that are adjacent to E and F, respectively. Let m and n be the chords different from l passing through A and B, respectivly. Let P and Q be the endpoints of m. Because L is 2–minimal, n intersects m once at most. Let R be an endpoint on n such that m does not intersect n between

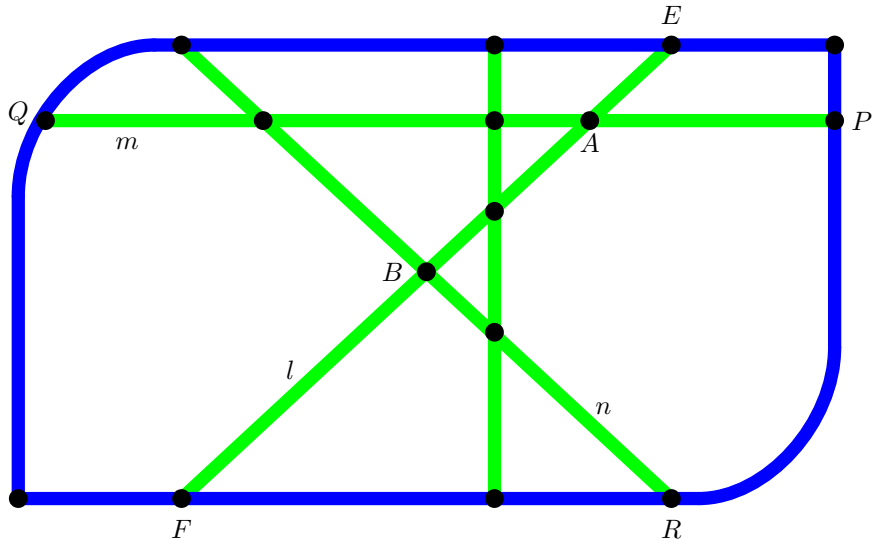
B and R.

Then the trails EQ, FR and PE on L do not overlap, and as L is of degree 2 or less, one of these trails are of degree 0. Let L^* be:

- the circuit consisting of the trail EQ on L, QA on m and AE on l, if EQ is of degree 0.
- the circuit consisting of the trail PE on L, EA on l and AP on m, if PE is of degree 0.
- the circuit consisting of the trail FR on L, RB on n and BF on l, if FR is of degree 0.

L^* is of degree 3, and it is 2–minimal as it is inside L. Because the degree 0-trails that L^* consist of don't intersect themselves and intersect each other at most once (because L is 2–minimal), L^* is simple. L^* shares an edge with L and has a different edge whose endpoints are corners in L^*. By lemma 5, there is a simple circuit of degree 3 in the interior of L^* that contains no chords that shares an edge with L, which completes the proof. □

Figure 20: Illustration for lemma 6.



Proof of Theorem 3. Let C be a generic plane curve with at least one crossing, and let L be a 2–minimal simple circuit of degree 1 or 2 in it (this uses lemma 4). From lemma 6 it follows that if the number of intersections between chords in the interior of L is not 0, L contains a simple circuit of degree 3 that contains no chords and shares an edge with L. You can use move 3 on this circuit to move a crossing out of L (see figure 21), thus reducing the number of chord

intersections in L, and this process can be repeated until none of the chords in L intersect. You can then use move 3 to remove each chord from the L (see figure 22). You can then use either reversed move 1 or reversed move 2 on L to reduce the number of crossings in C (see figure 23). As neither move 3 nor reversed move 2 increase the number of crossings, the total number of crossings in C has decreased. This process can then be repeated until C has no crossings, and thus is the trivial curve, which completes the proof.

□

Figure 21: Using move 3 to reduce the number of crossings in a 2–minimal simple circuit of degree 2.

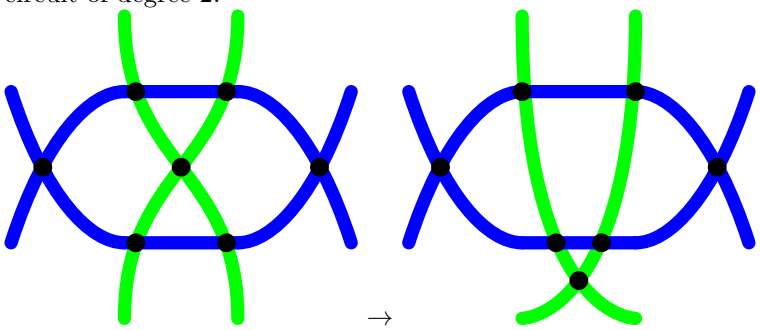


Figure 22: Using move 3 to reduce the number of chords in a 2–minimal simple circuit of degree 2 where no chords intersect in the circuit's interior.

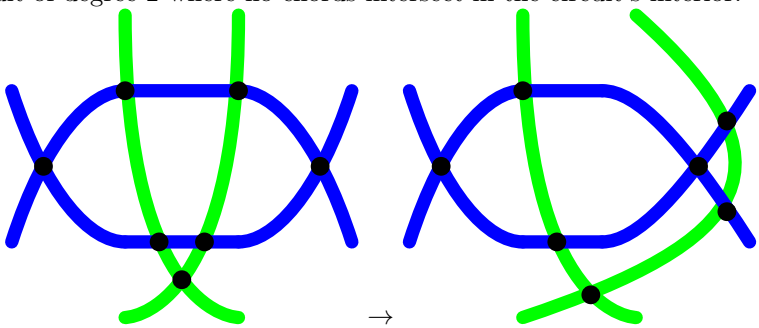
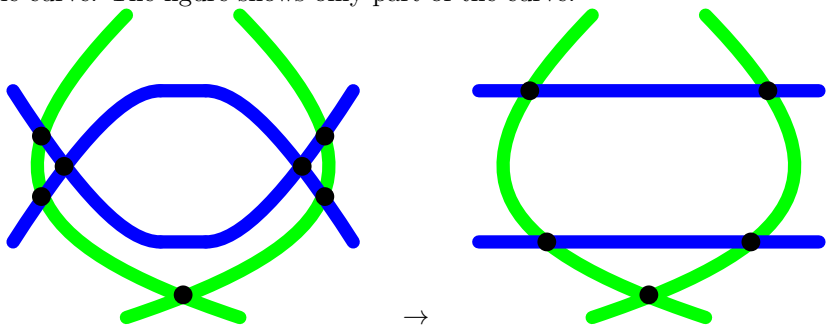


Figure 23: Using reversed move 2 to reduce the number crossings in a generic plane curve. The figure shows only part of the curve.



A Python script to calculate number of generic plane curves with n crossings

The code is based on theory from [Ghy17, p. 232-236]. Sample output is given in table 1.

```
n = int(input("n = "))

def connections(number, direction, before_plus, after_plus,
before_minus, after_minus):
    if direction == 'E':
        crosses = []
        if after_plus[1] == '+':
            crosses.append([after_plus[0], 'S'])
        else:
            crosses.append([after_plus[0], 'E'])
        if before_minus[1] == '+':
            crosses.append([before_minus[0], 'E'])
        else:
            crosses.append([before_minus[0], 'N'])
        return crosses
    elif direction == 'N':
        crosses = []
        if after_plus[1] == '+':
            crosses.append([after_plus[0], 'W'])
        else:
            crosses.append([after_plus[0], 'S'])
        if after_minus[1] == '+':
            crosses.append([after_minus[0], 'S'])
        else:
            crosses.append([after_minus[0], 'E'])
        return crosses
```

```
    elif direction == 'W':
        crosses = []
        if before_plus[1] == '+':
            crosses.append([before_plus[0], 'N'])
        else:
            crosses.append([before_plus[0], 'W'])
        if after_minus[1] == '+':
            crosses.append([after_minus[0], 'W'])
        else:
            crosses.append([after_minus[0], 'S'])
        return crosses
    elif direction == 'S':
        crosses = []
        if before_plus[1] == '+':
            crosses.append([before_plus[0], 'E'])
        else:
            crosses.append([before_plus[0], 'N'])
        if before_minus[1] == '+':
            crosses.append([before_minus[0], 'N'])
        else:
            crosses.append([before_minus[0], 'W'])
        return crosses

def validate_word(word):
    symbol_map = {}
    for i in range(len(word)):
        symbol = word[i]
        symbol_map[str(symbol[0]) + symbol[1]] = i
    checked = set()
    boundaries = 0
    for cross in range(1, n+1):
        for direction in ['E', 'N', 'W', 'S']:
            if (str(cross) + direction) not in checked:
                boundaries += 1
                current = [cross, direction]
                checked.add(str(current[0]) + current[1])
                while True:
                    before_plus = word[(symbol_map[str(current[0]) + '+'] - 1)%(2*n)]
                    after_plus = word[(symbol_map[str(current[0]) + '+'] + 1)%(2*n)]
                    before_minus = word[(symbol_map[str(current[0]) + '-'] - 1)%(2*n)]
                    after_minus = word[(symbol_map[str(current[0]) + '-'] + 1)%(2*n)]
                    new_connections = connections(current[0], current[1], before_plus,
                    after_plus, before_minus, after_minus)
                    str_1 = str(new_connections[0][0]) + new_connections[0][1]
                    str_2 = str(new_connections[1][0]) + new_connections[1][1]
                    if str_1 not in checked:
```

```
                        current = new_connections[0]
                        checked.add(str(current[0]) + current[1])
                    elif str_2 not in checked:
                        current = new_connections[1]
                        checked.add(str(current[0]) + current[1])
                    else:
                        break
    if boundaries - n == 2:
        return True
    return False

def count_words(n, word):
    if n == 0:
        return int(validate_word(word))
    count = 0
    for j in range(len(word) + 1):
        word1 = [[n, '+']]
        word1.extend(word[0:j])
        word1.append([n, '-'])
        word1.extend(word[j:])
        word2 = [[n, '-']]
        word2.extend(word[0:j])
        word2.append([n, '+'])
        word2.extend(word[j:])
        count += count_words(n-1, word1)
        count += count_words(n-1, word2)
    return count

print("Number of curves:", count_words(n, []))
```

Table 1: Sample ouput from the Python code.

#Crossings	#Curves
1	2
2	8
3	42
4	260
5	1796
6	13396

References

[Ghy17] É. Ghys. A singular mathematical promenade. ENS Éditions, Lyon, 2017.

[AB26] J.W. Alexander, G.B. Briggs. On types of knotted curves. Ann. of Math. 28 (1926/27), 562–586.

[Rei27] K. Reidemeister. Elementare Begründung der Knotentheorie. Abh. Math. Sem. Univ. Hamburg 5 (1927) 24–32.