Classification of Gap Functions with Degree 4

by

Yasaman Ahmadi

B.Sc., Shahid Beheshti University, 2018

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
Department of Mathematics
Faculty of Science

SIMON FRASER UNIVERSITY
Summer 2022

Declaration of Committee

Name:	**Yasaman Ahmadi**
Degree:	**Master of Science (Mathematics)**
Thesis title:	**Classification of Gap Functions with Degree 4**
Committee:	**Chair:** Katrina Honigs Assistant Professor, Mathematics
	Nathan Ilten Supervisor Associate Professor, Mathematics
	Imin Chen Committee Member Professor, Mathematics
	Marni Mishna Examiner Professor, Mathematics

Abstract

In algebraic geometry, gap functions are useful tools in the study of singularities of curves. It is thus desirable to have a classification of gap functions with arbitrary degrees. Gap functions with degrees 1 and 2 are known classically and Buczyński, Ilten, Ventura have classified gap functions with degree 3. In this thesis, building on the results of Buczyński, Ilten, Ventura, we present an algorithm that computes gap functions with arbitrary dimensions and degrees. After implementing the algorithm in Maple, we classify all 4 dimensional gap functions, which can help study all the curve singularities with delta invariant 4.

Keywords: Gap functions; curve singularities

To all women in science who,
in the face of all the challenges, work hard to be their best!

Acknowledgements

I respectfully acknowledge that this thesis was written on the unceded traditional territories of the Coast Salish peoples of the Squamish, Tsleil-Waututh, Musqueam and Kwikwetlem Nations.

I would like to thank Nathan, my supervisor. He guided me through every step of this project, and believed in me and gave me the opportunity to work under his supervision. Beyond the Math, he was always supportive and patient during some of the hardest days of my life.

I am grateful to Imin, my committee member, for his useful comments on this thesis. He was always understanding and concerned about my future as a graduate student. I am also thankful to Marni for her constructive comments on this thesis.

I am happy to have met Christie, whose kind words and warm support during my studies at SFU made her one of my best friends. Her presence at the defence session was heartening.

I thank Ahmad, as the first reader of this thesis, and I am gratefully indebted to his very valuable comments and help on this work, and his friendship during these years. Also, Abu and Saba, whose friendship was heartwarming and their presence in my life is one of the main reasons I could come so far. A big thanks goes to all of my friends.

Finally, I express my very profound gratitude to my parents, Maryam and Reza, and to my brother, Babak, for providing me with unfailing support and continuous encouragement throughout my life. This accomplishment would not have been possible without them. Thank you!

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

In algebraic geometry, the main objects of study are algebraic varieties, which are solution sets to systems of polynomial equations. Like in many other branches in math, after defining the objects that are to be studied, one is interested in classifying all those objects up to isomorphism. Thus, classification of varieties is a major problem. To make this problem more manageable, we can use invariants to focus on a smaller family of varieties. An invariant is a property of a variety that is preserved under isomorphism. For example dimension is an invariant of a variety. By considering varieties of dimension one, we arrive at the sub-problem of classifying curves.

When we study algebraic curves, the study of singularities is inevitable. A singularity is a point on a curve that is not smooth (see Definition 2.1.5). Even when we are interested in studying smooth curves, singular curves come into play. To see this, consider the geometric genus, another invariant of varieties. When the ground field is $\mathbb{C}$, the complex numbers, a smooth curve can be viewed as a two dimensional manifold over the real numbers. The geometric genus specifies how many holes this surface has. For example the real manifold corresponding to the projective line $\mathbb{P}^1_{\mathbb{C}}$ is a sphere with no holes and the geometric genus of $\mathbb{P}^1$ is thus 0. The geometric genus of a curve is a non-negative integer which lets us sub-divide the problem of classifying curves into the problem of classifying curves of a fixed genus g. Then, one can use moduli spaces to study this smaller family. For example, the moduli space M_g parameterizes smooth curves of genus g. Unfortunately, M_g is not compact, which prevents us from using those theorems that require compactness. A method for compactifying this space is the Deligne-Mumford compactification, which uses curves with nodal singularities [DM69].

Curve singularities have been studied extensively. For example, plane curves with nodal singularities have been studied extensively through moduli spaces called Severi varieties [DH88]. There is also the Clemens conjecture which states that a general quintic hypersurface in $\mathbb{P}^4$

will contain only finitely many smooth rational curves of a fixed degree and the singularities of the non-smooth ones can be quite constrained [Kat86; JK96; Cot05].

For plane curves, there are classical results regarding the classification of the singularities that can occur. A curve of degree d in $\mathbb{P}^2$ has arithmetic genus $d(d-1)/2$ (see [Har77, Exercise 7.2]). For $d \leq 5$ all possible configuration of singularities are known (see [Nam84]). There are partial results for $d = 6$ but a complete classification remains open.

In [IM21], the authors use classification of singularities with delta-invariant two to give an algorithm that determines which rational curves with arithmetic genus two admit a toric degeneration. Their work emphasizes the significance of classifying singularities with a given delta-invariant.

A method for studying the singular point on a curve is to consider the local ring of the curve at that point. The local ring of a point on a curve gives information about a Zariski open neighborhood of that point. However the Zariski topology is very coarse and an open set of the curve is the entire curve minus a finite number of points. To give more local information (e.g in a neighborhood of the point in the usual Euclidean topology) we can use a procedure called completion (see Definition 2.2.1). The isomorphism class of the completion of a local ring can be used as the definition of the singularity type of a point. Buczyński, Ilten and Ventura introduced gap functions as an invariant for the isomorphism class of the completion of the local rings [BIV20]. Thus, gap functions serve as tools for classifying singularities of curves.

Let $\mathbb{K}$ be an algebraically closed field. A gap function is a map coming from a subalgebra R of a finite product of rings of power series $\prod_{i=1}^{r} \mathbb{K}[[t_i]]$ (see Definition 2.4.1). To see how gap functions are related to singularities of curves, assume C is a curve and Q is a point on C. Let $\mathscr{O}_{C,Q}$ be the local ring of this point and R its completion. To get an invariant for the isomorphism class of R we can look at R relative to its normalization (see Definition 2.3.5) which is a product of rings of power series $S = \prod_{i=1}^{r} \mathbb{K}[[t_i]]$ (Proposition 2.3.6). We will see that the gap function corresponding to $R \subset S$ provides an invariant that describes how far R is from being equal to S. Here r is the number of branches of the curve at the singular point.

Example 1.1.1. Let $\mathbb{K} = \mathbb{C}$ be the field of complex numbers and let Y_1 be the affine plane curve given by the vanishing of $y^2 - x^2(x+1)$. See Figure 2.2a for a graph of the real points of Y_1. We will see in Example 2.1.7 that the origin $P = (0,0)$ is a singular point (a node) of Y_1. The completion of the local ring $\mathscr{O}_{Y_1,P}$ is isomorphic to

$$\hat{\mathscr{O}}_{Y_1,P} \cong \mathbb{K}[[x,y]]/(xy) \cong \mathbb{K}[[(t_1,0),(0,t_2)]] \subset \mathbb{K}[[t_1]] \times \mathbb{K}[[t_2]], \tag{1.1}$$

(See Example 2.2.6 for more details). Here $S := \mathbb{K}[[t_1]] \times \mathbb{K}[[t_2]]$ is the normalization of the completion of the local ring (see Proposition 2.3.6). The ring $\mathbb{K}[[(t_1,0),(0,t_2)]]$ is the power series ring in $(t_1,0),(0,t_2)$ where $\mathbb{K}$ is a subring by $c \mapsto (c,c)$. Thus elements of

$\mathbb{K}[[(t_1,0),(0,t_2)]]$ are of the form

$$\left(c+\sum_{i=1}^{\infty} a_i t_1^i, c+\sum_{i=1}^{\infty} b_i t_2^i\right) \in \mathbb{K}[[t_1]] \times \mathbb{K}[[t_2]], \quad \text{for some} \quad c, a_i, b_i \in \mathbb{K}.$$

Also, the second isomorphism in (1.1) is via the map that sends a constant c to (c,c), and $x \mapsto (t_1,0)$ and $y \mapsto (0,t_2)$. The gap function associated to the subalgebra $\mathbb{K}[[(t_1,0),(0,t_2)]] \subset S$ is the function $\lambda : \mathbb{Z}^2_{\geq 0} \to \mathbb{Z}_{\geq 0}$ where

$$\lambda(i,j) = \begin{cases} 0 & \text{if } i=0 \quad \text{or} \quad j=0, \\ 1 & \text{otherwise .} \end{cases}$$

We will see how to compute gap functions in Examples 2.4.5 and 2.4.6. For now we mention that if $\hat{\mathscr{O}}_{Y_1,P}$ had been equal to S, then the gap function λ would have been the zero function. The fact that λ is attaining non-zero values shows that $\hat{\mathscr{O}}_{Y_1,P}$ is not the full ring S and the higher the values λ attains, the further $\hat{\mathscr{O}}_{Y_1,P}$ is from being equal to S.

To specify a gap function it is enough to specify its values on $\mathbb{Z}^2_{>0}$ (the values at points with a zero component can be determined by the values on $\mathbb{Z}^2_{>0}$, see Corollary 2.5.2) and we usually write down these values as a table of integers. For example λ can be represented by the table

$$\begin{array}{|lll} \vdots & \vdots & \\ 1 & 1 & \dots \\ \mathbf{1} & 1 & \dots \end{array} \quad .$$

At the nodal singularity $(0,0) \in Y_1$ there are two distinct tangent directions and we say there are two branches and $r=2$. This is also reflected by the fact that the domain of λ is $\mathbb{Z}^r_{\geq 0} = \mathbb{Z}^2_{\geq 0}$.

The gap function λ provides an invariant for the isomorphism class of $\mathscr{O}_{Y_1,P}$ which can be used to distinguish a node from other singularity types. For example consider the affine curve Y_2 given by the vanishing of $y^2 - x^3$ (see Figure 2.2b for a graph of the real points of Y_2). The origin is a singular point (a cusp) whose singularity type is different from that of a node. This can be seen by comparing the gap function coming from a cusp (the gap function **1.1** in Table 2.1) to the gap function coming from a node (shown in the above table).

The *degree* of a gap function is the maximum value it attains, which is by design the same as the *delta-invariant* of the corresponding singularity. The delta-invariant is an invariant of a singular point that measures how bad the singularity is: As the value gets higher, the singularity becomes more complicated. The delta-invariant is related to another invariant of the curve, namely the arithmetic genus. It is a non-negative integer and the difference between the arithmetic genus and geometric genus measures how far a curve is from being

smooth. For example for a smooth curve the two genuses are equal. This difference between the two genuses is exactly equal to the sum of the delta-invariants of all singular points of the curve. See [Kun05, Chapter 14] for more details on the connection between the two genuses and delta-invariant of singularities. For singularities with delta-invariant one, two and three, the authors in [BIV20] show that gap functions completely determine the singularity types.

Classical results can be used to classify all gap functions with degree one and two. Buczyński, Ilten and Ventura [BIV20] have classified gap functions with degree three. The purpose of this thesis is to introduce an algorithm (see Section 3.2) which computes gap functions with any degree. By using an implementation of this algorithm, we can completely classify gap functions with degree 4 (see Theorem 4.2.1).

1.2 Approach

1.2.1 Generating gap functions

Given a positive integer δ, a *candidate gap function* with degree δ is a function $\lambda : \mathbb{Z}^r_{\geq 0} \to \mathbb{Z}$ whose maximum value is δ. The goal of the algorithm we give is to compute a set of candidate gap functions with degree δ such that the the set of gap functions with degree δ is a subset of the output. We will try to eliminate from the output those functions that are not gap functions as much as possible. The output of the algorithm for $\delta \leq 4$ are all the gap functions coming from curve singularities.

An important property of gap functions is that it can be completely specified by its values on a finite set of points. More specifically, for every gap function λ, there is a point $(a_1, \ldots, a_r) \in \mathbb{Z}^r_{\geq 0}$ such that λ is completely determined by its values on the set $\{(x_1, \ldots, x_r) : x_i \leq a_i\}$ (see Lemma 2.5.7). This allows us to store a gap function in a finite r-dimensional array in a computer. Moreover the a_i's are bounded from above and this bound depends solely on δ (see Lemmas 2.5.5 and 2.5.6). This means that we are searching for gap functions in a finite space of arrays. To eliminate tables that are not gap functions from the output as much as possible, we will use the following properties. Some of these properties were proved in [BIV20] and the rest are proved in Chapter 2:

1. Suppose $\lambda : \mathbb{Z}^r_{\geq 0} \to \mathbb{Z}$ is a gap function. Define

$$\begin{aligned} \lambda_H : \mathbb{Z}^{r-1}_{\geq 0} &\to \mathbb{Z} \\ (x_1, \ldots, x_{r-1}) &\mapsto \lambda(x_1, \ldots, x_{r-1}, 0), \end{aligned}$$

   and

$$\begin{aligned} \lambda_L : \mathbb{Z}_{\geq 0} &\to \mathbb{Z} \\ x &\mapsto \lambda(0, \ldots, 0, x). \end{aligned}$$

Then, λ_H and λ_L are gap functions with

$$\delta(\lambda) \geq \delta(\lambda_H) + \delta(\lambda_L) + 1. \tag{1.2}$$

(see Lemma 2.6.3). We act in reverse and start with an $r-1$-dimensional and a one-dimensional gap functions with degrees less than δ and try to assemble them into an r-dimensional gap function λ with degree δ.

2. From one entry of the table to the next, a gap function increases by at most one unit (see Lemma 2.5.1). Thus, we start from the entry $(1, \ldots, 1)$ and fill the table entry by entry until we reach an entry with the value δ. We stop and output the table.

3. Going from one entry to the next, often the values of the previous entries will force the gap function to increase by one (see Lemma 2.5.8). This property is called *upward propagation*, and thanks to this property many tables that are not gap functions will be eliminated.

We consider a table filled, once there is an entry with the value δ. As a final check, we test two conditions:

- The first condition is a technical one, that comes from a property of gap functions which we call the *semigroup property* (see Lemma 3.2.1). It will eliminate some of the tables in the output that are not gap functions.

- To minimize duplication, if under a permutation of the components, the table becomes equal to a previously obtained table, then the current table is is discarded. We call this the symmetry check.

The algorithm (see Chapter 3 for a complete description) will output the tables that pass the above two checks. We have implemented this algorithm in Maple [Map19].

At this point we have a list of tables that are candidates for being gap functions for curve singularities. The next step is to prove that each table is a gap function by manually finding the corresponding subalgebra $R \subset S = \prod_{i=1}^{r} \mathbb{K}[[t_i]]$ whose gap function is that table. In this thesis, we do this part for $\delta = 4$. We will see that there are 39 gap functions with $\delta = 4$. In this way, we classify all the gap functions with $\delta = 4$ and will prove the following theorem in Section 4.2.

Theorem 4.2.1. *The standard gap functions with degree 4, up to permutation of the coordinates, are those listed in Tables 4.1, 4.2, 4.3, and 4.4. Furthermore, Tables 4.5, 4.6, 4.7, 4.8, and 4.9 list the corresponding algebras $R \subset S$.*

1.2.2 Finding the subalgebras

Now we explain our method of finding R and S. Some of the tables are readily seen to be gap functions: These are the tables for which equality holds in Equation (1.2). We will prove that, using the upward propagation property, that every cell of the table is completely determined by the values of previous cells (see Proposition 2.6.6). Therefore, when equality holds, λ is uniquely determined by λ_H and λ_L. In this situation, we say λ is the *product* of λ_H and λ_L. The algebras R and S for λ are easily obtained from the corresponding algebras of λ_H and λ_L.

When equality does not hold in (1.2), the computation is not as easy. Here we follow the method used in [BIV20], which is look at entries where the gap function stays constant and to repeatedly use Lemmas 2.5.1 and 2.5.6 to produce generators for the subalgebra R of $S = \prod_{i=1}^{r} \mathbb{K}[[t_i]]$. In this way we will find the subalgebras for every table in the output, verifying that all of them are gap functions.

The structure of this thesis is as follows:

In Chapter 2, we recall basic concepts from algebraic geometry. We will then review the notions of completion and normalization of rings in commutative algebra. Then, we focus on gap functions and their relation to singularities of curves. After that, we talk about properties of gap functions that will be used in the algorithm. The first half of Chapter 3 presents the algorithm for computing gap function candidates in detail. The second half contains a proof of why the algorithm gives the correct output. The output of the algorithm for $\delta = 4$ is given in Tables in Chapter 4. Finally, in Chapter 4 we prove that each table in the output of the algorithm for $\delta = 4$ is a gap function. We include complete computations of the algebras $R \subset S$ for several cases. The tables in this chapter give the algebras $R \subset S$ for each output of the algorithm. Appendix A contains our implementation of the algorithm in Maple.

Chapter 2

Preliminaries

In this chapter, we review preliminary definitions and results. It helps if the reader is familiar with basic notions in algebraic geometry at the level of [Har77, Chapter 1]. Sections 2.1 through 2.3 review basic concepts such as varieties, local rings, smoothness, completion and normalization in algebraic geometry. Then, in Section 2.4 we give the definition of gap function and provide examples. After that, in Sections 2.5 and 2.6, we talk about the properties of gap functions. We provide proofs for properties that are new, and references for existing results. Finally, in Section 2.7 we will discuss the geometry of gap functions.

2.1 Basic notions

In this section we follow [Har77, Ch. I] and recall basic facts about affine varieties. Because we are interested in local behaviour of varieties, it is enough for our purposes to consider only affine varieties. Let $\mathbb{K}$ be an algebraically closed field. Let $\mathbb{A}^n = \mathbb{K}^n$ be the n-dimensional *affine space*. Closely related to $\mathbb{A}^n$ is the ring of polynomials $A = \mathbb{K}[x_1, \ldots, x_n]$ in n variables.

Definition 2.1.1. Let $f_1, f_2, \ldots, f_r \in A$ be polynomials. An *algebraic set* in $\mathbb{A}^n$ is a set of the form

$$V(f_1, \ldots, f_r) = \{(x_1, \ldots, x_n) \in \mathbb{A}^n : f_i(x_1, \ldots, x_n) = 0, 1 \leq i \leq r\}.$$

An algebraic set is *irreducible* if it is not the union of two proper algebraic subsets. An *affine variety* is an irreducible algebraic set.

The collection of algebraic subsets of $\mathbb{A}^n$ can be taken as the closed subsets of a topology on $\mathbb{A}^n$. This topology is called the *Zarisky topology*. An open subset in this topology is the complement of an algebraic set.

If $X \subset \mathbb{A}^n$ is an affine variety, the dimension of X, denoted by $\dim X$, is by definition the supremum of all integers n such that there is a chain of length n of affine varieties X_i contained in X:

$$X_0 \subset X_1 \subset \ldots \subset X_n = X.$$

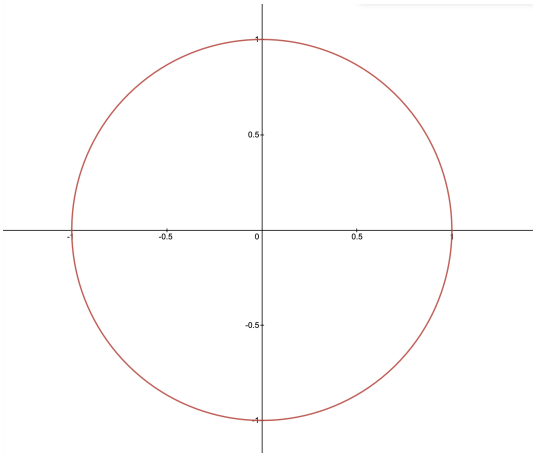


Figure 2.1: The real points on the plane curve $x^2 + y^2 - 1 = 0$ in Example 2.1.3.

One can show that $\dim \mathbb{A}^n = n$ [Har77, Proposition I.1.9] and for two varieties $X \subset Y$, $\dim X \leq \dim Y$ [Har77, Exercise I.1.10]. Hence, an affine variety in $\mathbb{A}^n$ has dimension at most n. In this thesis, when we draw a variety, we implicitly assume that $\mathbb{K} = \mathbb{C}$ and that we are plotting the real points of that variety.

Definition 2.1.2. A curve is a variety of dimension 1.

Example 2.1.3. Let X be the curve defined by $X = V(x^2+y^2-1) \subset \mathbb{A}^2$. Figure 2.1 shows a plot of X.

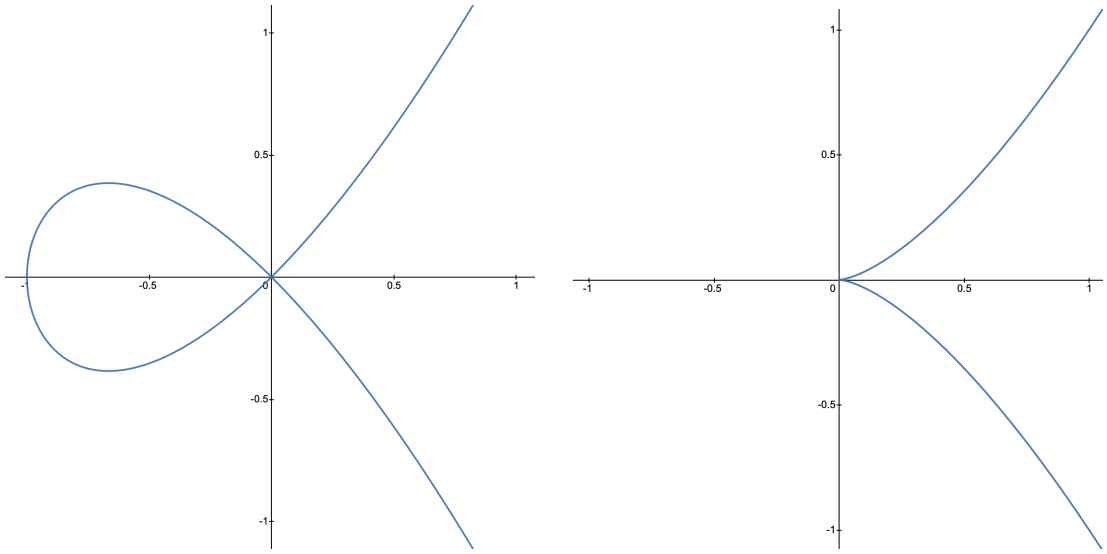
Example 2.1.4. The varieties $Y_1 = V(y^2 - x^2(x+1))$ and $Y_2 = V(y^2 - x^3)$ are other examples of plane curves. Figure 2.4 shows these curves. There is a major difference between the curve in Example 2.1.3 and the ones in Example 2.1.4. Every point on the curve X is a smooth point whereas, the curves Y_1 and Y_2 have a point (the origin) where the curve either intersects itself (in Y_1) or the point is a corner point (in Y_2). We call these points singular points of the curves. In the next definition, we make the notion of smoothness precise.

Recall that an additive subgroup I of a ring A is called an *ideal* if for every $a \in A$ and $x \in I$, we have $ax \in I$. The ideal *generated by* a subset of A is the smallest ideal of A containing that subset. For a variety $X \subset \mathbb{A}^n$, the *ideal of* X is the ideal of $A = \mathbb{K}[x_1, \ldots, x_n]$ generated by

$$\{f \in A : \ f(P) = 0 \quad \text{for all} \quad P \in X\}.$$

A famous theorem in commutative algebra, namely Hilbert's basis theorem [Eis95, Theorem 1.2] states that the ideal of a variety is always finitely generated.

Definition 2.1.5 ([Har77, Chapter I.2])**.** Let $X \subset \mathbb{A}^n$ be an affine variety of dimension r. Let $f_1, \ldots, f_t \in A = \mathbb{K}[x_1, \ldots, x_n]$ be polynomials which generate the ideal of X. Let $P \in X$ be a point, with coordinates $P = (a_1, \ldots, a_n)$. The point P is *nonsingular* if the rank of the matrix $||(\partial f_i/\partial x_j)(a_1, \ldots, a_n)||$ is $n - r$. Otherwise, the point P is *singular*.



(a) the nodal curve $Y_1 : y^2 - x^2(x+1) = 0$ (b) the cuspidal curve $Y_2 : y^2 - x^3 = 0$

Figure 2.2: The plane curves in Example 2.1.4 with singularities at $(0, 0)$

Example 2.1.6. Let X be as in Example 2.1.3. The ideal of X is generated by $f = x^2 + y^2 - 1$. Then

$$\begin{bmatrix} \partial f/\partial x & \partial f/\partial y \end{bmatrix} = \begin{bmatrix} 2x & 2y \end{bmatrix},$$

has rank 1 for every point on X. This shows that every point of X is smooth (see Figure 2.1).

Example 2.1.7. Let Y_1 be as in Example 2.1.4. The ideal of Y_1 is generated by $f = y^2 - x^2(x+1)$. Since the matrix

$$\begin{bmatrix} \partial f/\partial x & \partial f/\partial y \end{bmatrix} = \begin{bmatrix} -3x^2 - 2x & 2y \end{bmatrix},$$

has rank zero at the point $(0, 0)$, this point is a singular point of the curve. This point is the origin in Figure 2.4. A similar computation shows that $(0, 0)$ is a singular point of Y_2.

To study the singular points of a variety, we consider the local ring of the variety at a point.

Definition 2.1.8 ([Har77, Section I.3])**.** Let $X \subset \mathbb{A}^n$ be a variety and suppose $P \in X$. A function $f : X \to \mathbb{K}$ is *regular* at P, if there is an open neighbourhood U of P and polynomials $g, h \in \mathbb{K}[x_1, \ldots, x_n]$, such that h does not vanish on U and $f = \frac{g}{h}$. We say f is regular on X, if it is regular at every point of X.

The next definition introduces the local ring of a variety at a point.

Definition 2.1.9 ([Har77, Section I.3])**.** Let X be a variety and $P \in X$. The *local ring of P on X*, denoted by $\mathcal{O}_{X,P}$, is the set of all pairs (U, f), where U is an open subset of X

containing P and f is a regular function on U. We identify two pairs (U, f) and (V, g), if on $U \cap V$, $f = g$.

The local ring $\mathscr{O}_{X,P}$ has the unique maximal ideal

$$M = \{(U, f) \in \mathscr{O}_{X,P} : f(P) = 0\},$$

since if $(U, f) \notin M$, i.e. $f(P) \neq 0$, then (U, f) is a unit with inverse $(U \backslash V(f), 1/f)$. A basic result from algebraic geometry tells us what the local ring of a variety at a point looks like. To state this result we need the notion of localization of rings.

Definition 2.1.10. Let R be a ring and $U \subset R$ a subset containing 1 that is closed under multiplication. The *localization of R with respect to U*, denoted by $U^{-1}R$, is the set of equivalence classes of formal fractions $\frac{r}{u}$ with $r \in R, u \in U$. Two such fractions $\frac{r_1}{u_1}, \frac{r_2}{u_2}$ are equivalent if there exists $u' \in U$ such that $u'(r_1u_2 - r_2u_1) = 0$.

One can prove that the equivalence classes of fractions $U^{-1}R$ in the above definition is a ring with 1 [AM69, Section 3]. An important example of a multiplicatively closed set U is $U = R \backslash I$ where $I \subset R$ is a prime ideal. In this situation we denote the localization of R with respect to $R \backslash I$ by R_I.

Theorem 2.1.11 ([Har77, Section I.3])**.** *Let X be a variety and suppose I is the ideal of X. For any point $Y \in X$, let $M_P \subset A(X) = \mathbb{K}[x_1, \ldots, x_n]/I$ be the maximal ideal generated by the set of polynomials $f \in A(X)$ such that $f(P) = 0$, then $\mathscr{O}_P$ is the localization $A(X)_{M_P}$.*

Example 2.1.12. Let Y_2 be as an Example 2.1.4 and consider $P = (0, 0)$. Then, $A(Y_2) = \mathbb{K}[x, y]/(y^2 - x^3)$ and

$$\mathscr{O}_{Y_2,P} \cong \left(\mathbb{K}[x, y]/(y^2 - x^3)\right)_{(x,y)}.$$

2.2 Completion

Let X be a variety and $P \in X$. The local ring $\mathscr{O}_{X,P}$ contains information about a neighbourhood of P. But in the Zariski topology, a neighbourhood of a point almost covers the variety X and is too big. For example, if X is a curve, a Zariski open neighbourhood of P is X minus a finite number of points. Thus, the local ring $\mathscr{O}_{X,P}$ gives information about almost all of X. To study the local behaviour of a point $P \in X$, we can use a procedure called completion, which takes a ring R and produces a related ring $\hat{R}$.

To illustrate the kind of information we can get from $\hat{R}$, suppose for the moment that we are working over $\mathbb{C}$, the field of complex numbers. Over $\mathbb{C}$, we have two topologies, the Zariski topology and the usual Euclidean topology. If we are interested in the local behaviour of P, we might want to consider a neighbourhood of P in Euclidean topology. By passing from $\mathscr{O}_{X,P}$ to the completion $\hat{\mathscr{O}}_{X,P}$, we are in a sense achieving this. The current section defines completion and presents some of its properties.

Definition 2.2.1 ([Eis95, Chapter 7])**.** Let R be a ring and let $M \subset R$ be an ideal. The *completion* of R with respect to the ideal M, denoted $\hat{R}_M$, is the ring

$$\hat{R}_M = \left\{ (g_1, g_2, \ldots) \in \prod_{i=1}^{\infty} R/M^i : g_j \equiv g_i \,(\mathrm{mod}\, M^i) \text{ for all } j > i \right\}.$$

If M is understood from the context, we denote the completion simply by $\hat{R}$.

To see what the intuition behind completion is, consider the case where the ground field is $\mathbb{C}$, the complex numbers. Any rational function (in one or more variables) can be considered as an analytic function by considering its power series expansion around a point. The power series expansion is a powerful tool in the analytic setting (i.e. when $\mathbb{K} = \mathbb{C}$) but it is not applicable in the algebraic setting (i.e. when $\mathbb{K}$ is an arbitrary algebraically closed field). We can still use the formal power series through completion. By using the completion of the local ring of a variety at a point, we can gain more local information compared to the data the local ring provides. The next example makes the connection to formal power series more clear.

Example 2.2.2. Let $R = \mathbb{K}[x]$ and $M = (x)$. We claim that the completion $\hat{R}_M$ is isomorphic to $\mathbb{K}[[x]]$, the ring of formal power series in one variable. To see this consider the ring homomorphism

$$\begin{aligned} &\phi : \mathbb{K}[[x]] \to \hat{R}_M \\ &\sum_{i=0}^{\infty} a_i x^i \mapsto (a_0, a_0 + a_1 x, a_0 + a_1 x + a_2 x^2, \ldots). \end{aligned}$$

This map is obviously injective. To prove surjectivity, note that each element $g \in \hat{R}_M$ has a unique representative $(g_1, g_2, \ldots)$ where for every $i \geq 1$, g_i is a polynomial of degree less than i, and $g_i - g_{i-1}$ is either 0 or a monomial of degree $i-1$. Let a_{i-1} be the coefficient of this monomial (or 0, if $g_i - g_{i-1} = 0$). Then, $\sum_{i=0}^{\infty} a_i x^i \in \mathbb{K}[[x]]$ maps to g.

Example 2.2.3. Let $R = \mathbb{K}[x_1, \ldots, x_n]$ and $M = (x_1, \ldots, x_n)$. Similar to Example 2.2.2, one can show that $\hat{R}_M \cong \mathbb{K}[[x_1, \ldots, x_n]]$.

When the ring R is the local ring $\mathscr{O}_{X,P}$, we will always apply completion with respect to its unique maximal ideal. In this way, the notation $\hat{\mathscr{O}}_{X,P}$ is unambiguous.

Theorem 2.2.4 ([Har77, Theorem 5.4A, 5.5A])**.** *Let X be a variety of dimension n and let $P \in X$ be a smooth point. If $\hat{\mathscr{O}}_{X,P}$ is the completion of the local ring $\mathscr{O}_{X,P}$ with respect to its maximal ideal, then*

$$\hat{\mathscr{O}}_{X,P} \cong \mathbb{K}[[x_1, \ldots, x_n]].$$

According to this theorem, we cannot distinguish between smooth points on varieties of the same dimension by looking at the completion of their local rings.

Definition 2.2.5 ([Har77, Section I.5])**.** Let X, Y be two varieties. We say the point $P \in X$ and $Q \in Y$ are *analytically isomorphic*, if there is an isomorphism $\hat{\mathcal{O}}_{X,P} \cong \hat{\mathcal{O}}_{Y,Q}$ as $\mathbb{K}$-algebras.

By Theorem 2.2.4, any two smooth points on varieties of the same dimension are analytically isomorphic.

Example 2.2.6 ([Har77, Example I.5.6.3])**.** Consider the nodal curve Y_1 in Example 2.1.4. The curve is defined by the vanishing of $f(x, y) = y^2 - x^2(x + 1)$. Although $f(x, y)$ is irreducible in $\mathbb{K}[x, y]$ (and thus the nodal curve is irreducible), we show that in $\mathbb{K}[[x, y]]$, this polynomial factors as $f = gh$ where

$$g = y + x + g_2 + g_3 + \cdots,$$
$$h = y - x + h_2 + h_3 + \cdots,$$

where g_i, h_i are homogeneous polynomials of degree i. We can inductively find g_i, h_i. For example, for $i = 2$, by looking at the degree 3 parts of $f = gh$, we get

$$-x^3 = (y - x)g_2 + (y + x)h_2.$$

Because $(y - x, y + x) = (x, y)$, we can solve the above equation for g_2, h_2. Next, we find g_3, h_3 by looking at the degree 4 parts of $f = gh$ and continue in this manner to get all g_i, h_i. This shows that from the point of view of power series, the polynomial $f(x, y)$ is reducible.

Now let us look at the local ring of Y_1 at the origin $P = (0, 0)$. It is

$$\mathcal{O}_{Y_1,P} = \mathbb{K}[x, y]_{(x,y)}/(f).$$

The completion R of the local ring with respect to its maximal ideal will be

$$R \cong \mathbb{K}[[x, y]]/(f).$$

Because f factors in $\mathbb{K}[[x, y]]$ as $f = gh$ where g, h have linear terms, there is an automorphism of $\mathbb{K}[[x, y]]$ that maps $g \mapsto x$ and $h \mapsto y$, i.e.

$$R \cong \mathbb{K}[[x, y]]/(xy).$$

This corresponds to the fact that in a small neighbourhood (in the sense of the usual Euclidean topology) of the origin, the nodal curve is the union of two branches (see Figure 2.2a). That is the singular point of Y_1 is analytically isomorphic to the singular point of two intersecting lines defined by $xy = 0$.

We now define the singularity type of a point on a curve.

Definition 2.2.7. Let C be a curve. The *singularity type* of a point P on C is the isomorphism class of $\hat{\mathcal{O}}_{C,P}$.

We see that for two curves C, C', the point $P \in C$ and $P' \in C'$ have the same singularity type if and only if they are analytically isomorphic.

2.3 Normalization

In this section we review a notion from commutative algebra called *normalization*. Normalization helps us in producing invariants for the local ring $\hat{\mathcal{O}}_{C,P}$.

Definition 2.3.1 ([AM69, Ch. 5])**.** Let R be a subring of S. We say $s \in S$ is *integral* over R if there exists a monic $f(x) \in R[x]$ such that $f(s) = 0$.

Example 2.3.2. Let $R = \mathbb{K}[t^2, t^3] \subset S = \mathbb{K}[t]$. Then $t \in S$ is integral over R because it is the root of $f(x) = x^2 - t^2$.

Example 2.3.3. Let $R = \mathbb{K}[t^2, t^3] \subset S = \mathbb{K}[t]$. Then $\frac{1}{t} \in S$ is not integral over R. To see this suppose $f(x) = x^n + a_1 x^{n-1} + \ldots + a_n \in R[x]$ is a polynomial with $f(\frac{1}{t}) = 0$. Then

$$\left(\frac{1}{t}\right)^n + a_1 \left(\frac{1}{t}\right)^{n-1} + \ldots + a_n = 0.$$

By multiplying by t^n, we get

$$1 + t(a_1 + ta_2 + \ldots + a_n t^n) = 0,$$

implying that t divides 1, a contradiction.

Lemma 2.3.4. *For two rings $R \subset S$, the set $C = \{s \in S : s \text{ is integral over } R\}$ is a subring of S containing R.*

Proof. See [AM69, Corollary 5.3]. □

We recall that for a ring R, the *total quotient ring* of R is the localization of R with respect to the set of non-zerodivisors of R. The natural map from R to the total quotient ring given by $r \mapsto \frac{r}{1}$ is injective (see for example [Eis95, Section 2.1]). When R is an integral domain, then $U = R \backslash \{0\}$ is the set of non-zerodivisors and the localization $U^{-1}R$ is a field, called the *field of fractions* of R.

Definition 2.3.5. The ring C in the Lemma 2.3.4 is called the integral closure or normalization of R in S. If $R = C$, we say R is integrally closed or normal. If for a ring R, S is not specified, we take S to be the total quotient ring of R.

Proposition 2.3.6 ([Kun05, Theorem 16.14]). *Let C be a curve and $P \in C$. Then the normalization of $\hat{\mathcal{O}}_{C,P}$ is isomorphic to*

$$\prod_{i=1}^{r} \mathbb{K}[[t_i]],$$

for some $r \in \mathbb{Z}_{>0}$. We call r the number of branches of C at P.

Example 2.3.7. Consider the nodal curve Y_1 in Example 2.1.4 given by $f(x,y) = y^2 - x^2(x+1)$. Let $P = (0,0)$ be the point at the origin. We saw in Example 2.2.6 that the completion R of the local ring $\mathcal{O}_{Y_1,P}$ with respect to its maximal ideal is

$$R \cong \mathbb{K}[[x,y]]/(xy).$$

In Example 1.1.1 we saw the isomorphism

$$\mathbb{K}[[x,y]]/(xy) \cong \mathbb{K}[[(t_1,0),(0,t_2)]] \subset \mathbb{K}[[t_1]] \times \mathbb{K}[[t_2]],$$

and it is straightforward to show that $\mathbb{K}[[t_1]] \times \mathbb{K}[[t_2]]$ is both integrally closed and integral over $\mathbb{K}[[(t_1,0),(0,t_2)]]$. Therefore, the normalization of R is isomorphic to the product of two rings of power series. Here the curve has two branches at the origin (with tangent lines $y - x = 0$ and $y + x = 0$, the linear terms of g, h). Similarly, for the cuspidal curve Y_2 in Example 2.1.4, the completion of the local ring at the origin is isomorphic to $\mathbb{K}[[t^2, t^3]]$ with normalization $\mathbb{K}[[t]]$. The curve Y_2 has one branch with a double tangent line $y = 0$.

2.4 Gap functions

Previously, we saw that for a curve C and a point $P \in C$, we can take the isomorphism class of $\hat{\mathcal{O}}_{C,P}$ as the singularity type of the point P. In this section, we present the concept of gap function which is an isomorphism invariant of $\hat{\mathcal{O}}_{C,P}$. We saw in Section 2.3 that S, the normalization of $\hat{\mathcal{O}}_{C,P}$, is of the form

$$S = \prod_{i=1}^{r} \mathbb{K}[[t_i]],$$

where $r \in \mathbb{Z}_{>0}$.

Definition 2.4.1 ([BIV20, See Section 2]). For any $\mathbb{K}$-*subalgebra* $R \subset S$, the *gap function* of R in S is the map $\lambda_R : \mathbb{Z}^r_{\geq 0} \to \mathbb{Z}_{\geq 0}$ with

$$\lambda_R(\alpha) = \dim S/\left(R + \langle t_1^{\alpha_1}, \ldots, t_r^{\alpha_r} \rangle\right),$$

where $\langle \bullet \rangle$ denotes the ideal in S generated by $\bullet$, while the quotient is of vector spaces.

The *degree* of the gap function λ_R is

$$\delta(\lambda_R) = \sup_{\alpha \in \mathbb{Z}^r_{\geq 0}} \lambda_R(\alpha).$$

A gap function is standard if $\lambda(e_i) = 0$ for any i, and $\lambda(e_1 + \ldots + e_r) = r - 1$, where e_i is the i-th standard basis vector of $\mathbb{Z}^r$.

We note that since the quotient of vector spaces $S/\langle t_1^{\alpha_1}, \ldots, t_r^{\alpha_r} \rangle$ is finite-dimensional, the dimension in the above definition is always finite. Moreover, in applications, R is the completion of a local ring and S its normalization and in this case the degree of the corresponding gap function is also finite.

Remark 2.4.2. In Definition 2.4.1, $t_i^{\alpha_i}$ denotes the n-tuple

$$(0, \ldots, 0, t_i^{\alpha_i}, 0, \ldots, 0),$$

in S, where $t_i^{\alpha_i}$ appears in the i-th position. With this notation, we can write for example

$$(t_1 + t_2)(t_1 + 2t_2) = (t_1^2 + 2t_2^2).$$

Note that t_i^0 then denotes the n-tuple with 1 in the i-th position and zero elsewhere.

Remark 2.4.3. In [BIV20], the authors define a gap function for any $\mathbb{K}$-vector subspace $R \subset S$. Because we will only be dealing with subalgebras of S, in Definition 2.4.1, we defined gap functions only for subalgebras.

Notation 2.4.4 ([BIV20, Section 2])**.** There is a standard discrete valuation map $\nu : \mathbb{K}[[t]] \to \mathbb{Z} \cup \{\infty\}$, which is defined by

$$\nu : \sum_{i=0}^{\infty} a_i t^i \mapsto \min\{i \in \mathbb{Z}_{\geq 0} : a_i \neq 0\}.$$

Over the ring $S = \prod_{i=1}^r \mathbb{K}[[t_i]]$, there are r discrete valuations, $\nu_i : S \to \mathbb{Z} \cup \infty$. These are obtained by projecting S to its i-th factor and taking the standard discrete valuation on $\mathbb{K}[[t_i]]$. We also obtain the following map on a subalgebra $R \subset S$,

$$\nu : R \to (\mathbb{Z} \cup \infty)^r,$$
$$(f_1, \ldots, f_r) \mapsto (\nu_1(f_1), \ldots, \nu_r(f_r)).$$

We denote by Σ the image $\nu(R)$. It is easy to see that if R is a ring, then Σ is a semigroup.

Given an element $\alpha \in \mathbb{Z}^r_{\geq 0}$ and an index $1 \leq i \leq r$, α_i denotes the i-th coordinate of α. We set $|\alpha| = \Sigma_{i=1}^r \alpha_i$. The element $e_i \in \mathbb{Z}^r$ denotes the i-th vector of the standard basis. Given some element $\alpha \in \mathbb{Z}^r_{\geq 0}$ and an index $1 \leq i \leq r$, we will say that $\alpha[i]$ belongs to or

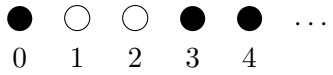


Figure 2.3: The picture of the gap function in Example 2.4.5

is contained in Σ if there exists an element $\alpha' \in \Sigma$ such that $\alpha_j \leq \alpha'_j$ for all $1 \leq j \leq r$, and $\alpha_i = \alpha'_i$. We will also refer to $\alpha[i]$ as an element of Σ. This notation will facilitate our discussion when we have a gap function and we want to find the generators of the corresponding subalgebra.

Example 2.4.5. We would like to compute the gap function of the subalgebra $R = \mathbb{K}[[t^3, t^4, t^5]] \subset S = \mathbb{K}[[t]]$. To compute $\lambda(\alpha)$ when $\alpha = 0$ we note that the ideal $\langle 1 \rangle \subset S$ is the entire ring S, hence $R + \langle 1 \rangle = S$ and

$$\lambda(0) = \dim S/\left(R + \langle 1 \rangle\right) = \dim S/S = 0.$$

When $\alpha = 1$, the ideal $\langle t \rangle \subset S$ is the set of all power series without the term t^0. Because R contains the constants (i.e. elements of $\mathbb{K}$), we have $R + \langle t \rangle = S$ and

$$\lambda(1) = \dim S/\left(R + \langle t \rangle\right) = \dim S/S = 0.$$

For $\alpha = 2$ since $R + \langle t^2 \rangle = \mathbb{K}[[t^2, t^3]]$ lacks the basis vector t,

$$\lambda(2) = \dim S/\left(R + \langle t^2 \rangle\right) = \dim S/\left(\mathbb{K}[[t^2, t^3]]\right) = 1.$$

Similarly, because for $\alpha \geq 3$, $R + \langle t^\alpha \rangle = R$ lacks the basis vectors t and t^2, we have

$$\lambda(\alpha) = \dim S/\left(R + \langle t^\alpha \rangle\right) = \dim S/R = 2.$$

Recall that the degree of the gap function λ is the largest value λ attains, which in this case is 2. This has a natural explanation, because two basis elements $t, t^2 \in S$ are missing in R. The gap function counts these missing basis elements. We can visualize the gap function λ_R as in Figure 2.3. In this figure, a circle at integer α is filled if the basis element $t^\alpha \in S$ is in R (not missing), and an empty circle shows that the basis element t^α is missing in R.

We always show gap functions as a table of values. For instance, the gap function λ_R in example 2.4.5 is

0	**1**	**2**	2	...

In the above table, the values of the gap function $\lambda_R(\alpha)$ are shown for $\alpha \geq 1$. The first time a value shows up in the table is marked with a blue colour. We note that the values in the table are increasing and they stabilize once the degree of the gap function is attained.

Example 2.4.6. Let $R = \mathbb{K}[[t_1 + t_2, t_2^2]] \subset S = \mathbb{K}[[t_1]] \times \mathbb{K}[[t_2]]$. In order to compute the gap function λ_R first we find some elements of S that are not in R.

1. First, we want to show that $\alpha_2 t_2^2 + \alpha_3 t_2^3 + \ldots \in R$ for every $\alpha_i \in \mathbb{K}$: By multiplying $(t_1 + t_2) \in R$ by $\alpha_3 t_2^2 + \alpha_5 t_2^4 + \ldots \in R$, we have $\alpha_3 t_2^3 + \alpha_5 t_2^5 + \ldots \in R$, then if we add it to $\alpha_2 t_2^2 + \alpha_4 t_2^4 + \ldots \in R$, we have $\alpha_2 t_2^2 + \alpha_3 t_2^3 + \alpha_4 t_2^4 + \ldots \in R$. This means $\lambda_R(k, 2) = \lambda_R(k, i)$ for $i > 2$ and $k \geq 0$.

2. We show that $\beta_2 t_1^2 + \beta_3 t_1^3 + \ldots \in R$: If we substitute $(t_1 + t_2)$ for t_1 in the above expression, we have

$$\beta_2(t_1 + t_2)^2 + \beta_3(t_1 + t_2)^3 + \ldots = \beta_2(t_1^2 + t_2^2) + \beta_3(t_1^3 + t_2^3) + \ldots.$$

   By (1.), series that only have t_2 to the power of 2 and more are in R, and $(t_1 + t_2)^i$ for $i \geq 2$ is also in R. If we subtract them, then we can conclude that t_1 to the power of 2 and more is also in R. Hence, $\lambda_R(2, k) = \lambda_R(i, k)$ for $i > 2$ and $k \geq 0$.

3. Since $(0, t_2^2) \in R$, $(t_1^2, 0) \in R$, we have $(0, t_2^2) + (t_1^2, 0) = (t_1^2, t_2^2) \in R$.

4. At this point, the points $(\beta_0 + \beta_1 t_1, \alpha_0 + \alpha_1 t_2)$ are left. In order to find $\dim S/R$, we have to find a basis. For the basis we have two conditions: Firstly, elements of the basis generates all the elements $(\beta_0 + \beta_1 t_1, \alpha_0 + \alpha_1 t_2)$. Secondly, they are linearly independent.

   We know that $(1, 0), (t_1, 0), (0, 1), (0, t_2)$ generates all the elements $(\beta_0 + \beta_1 t_1, \alpha_0 + \alpha_1 t_2)$, but some of them may be linearly dependent.

   - We want to show that $(t_1, 0), (0, t_2)$ are linearly dependent, i.e. a combination of them has to be zero in S/R, so their combination has to be in R.

$$(t_1, 0) + (0, t_2) = (t_1, t_2) = (t_1 + t_2) \in R.$$

     Thus, we have to eliminate one of them for example $(0, t_2)$.

   - We want to show that $(1, 0), (0, 1)$ are linearly dependent too. R is a $\mathbb{K}$-algebra generated by $t_1 + t_2$ and t_2^2. It means that a copy of $\mathbb{K}$ is in R, i.e. R has the elements of the form (C, C), where C is a constant. Thus, $(1, 1)$ is in R. Since $(1, 1) - (1, 0) = (0, 1)$, $(1, 0)$ and $(0, 1)$ are also linearly dependent. Now, we can eliminate one of them for example $(0, 1)$.

     By these two observations, $(1, 0), (t_1, 0)$ is a basis for $(\beta_0 + \beta_1 t_1, \alpha_0 + \alpha_1 t_2)$. Therefore, the general elements in S/R can be written $(\beta_0 + \beta_1 t_1, 0)$. Hence, $\dim S/R = 2$, and we cannot generate $(\beta_0 + \beta_1 t_1, 0)$ by $t_1 + t_2$ and t_2^2.

5. We have $(t_1, 0)$ in basis, and $(0, t_2^3) \in R$, $(t_1, 0) + (0, t_2^3) = (t_1, t_2^3)$. Thus, $(t_1, t_2^3) \in R$.

By finding the elements that are in R, we now know the elements that are not in R: $(1,0),(t_1,0)$.

Now, we can find the gap function for $R=\mathbb{K}[[t_1+t_2,t_2^2]]$.

$$\lambda(0,0)=\dim S/\left(R+\langle t_1^0,t_2^0\rangle\right)=\dim S/\left(R+\langle 1\rangle\right)=0,$$
$$\lambda(1,0)=\dim S/\left(R+\langle t_1^1,t_2^0\rangle\right)=\dim S/\left(R+\langle (t_1,0),(0,1)\rangle\right)=0,$$

because we are adding the elements that are not in R to R, we have everything in the denominator. Therefore, the dimension of the quotient is zero. Similarly,

$$\lambda(0,1)=\dim S/\left(R+\langle t_1^0,t_2^1\rangle\right)=\dim S/\left(R+\langle (1,0),(0,t_2)\rangle\right)=0,$$
$$\lambda(1,1)=\dim S/\left(R+\langle t_1,t_2\rangle\right)=\dim S/\left(R+\langle (t_1,0),(0,t_2)\rangle\right)=\#\{(1,0)\}=1,$$
$$\lambda(2,0)=\dim S/\left(R+\langle t_1^2,t_2^0\rangle\right)=\dim S/\left(R+\langle (t_1^2,0),(0,1)\rangle\right)=\#\{(t_1,0)\}=1,$$
$$\lambda(0,2)=\dim S/\left(R+\langle t_1^0,t_2^2\rangle\right)=\dim S/\left(R+\langle (1,0),(0,t_2^2)\rangle\right)=\#\{(t_1,0)\}=1,$$

Since $(t_1^2,0)$ and $(0,t_2^2)$ are in R, they are zero in S/R.

$$\lambda(2,2)=\dim S/\left(R+\langle t_1^2,t_2^2\rangle\right)=\dim S/\left(R+\langle (t_1^2,0),(0,t_2^2)\rangle\right)=\#\{(1,0),(t_1,0)\}=2.$$

Therefore, $\delta(\lambda_R)=\sup_\alpha \lambda_R(\alpha)=2$. The gap function λ_R is

$\vdots$	$\vdots$	
1	**2**	$\dots$
1	1	$\dots$

Example 2.4.7. Let C_1 and C_2 be two curves. Pick smooth points $P_1\in C_1$ and $P_2\in C_2$. Then by Theorem 2.2.4 the rings $\hat{\mathscr{O}}_{C_1,P_1}$ and $\hat{\mathscr{O}}_{C_1,P_1}$ are both isomorphic to the power series ring in one variable $\mathbb{K}[[t]]$. Becuase the normalization of $\mathbb{K}[[t]]$ is itself, we conclude that the gap functions corresponding to the singularity types of P_1 and P_2 are equal to the zero function $\lambda:\mathbb{Z}_{\geq 0}\to\mathbb{Z}_{\geq 0}$. This illustrates how gap functions are invariants for the isomorphism class of the completion of local rings.

2.5 Known properties of gap functions

The algorithm that we will give in next chapter for computing gap functions uses many properties of gap functions. In this section we will review those properties that have been proved in [BIV20]. We have chosen not to include some of the proofs for two reasons: Firstly, they are available in [BIV20] and secondly, the proofs are not essential in the description of our algorithm. Throughout this section, λ is the gap function of some subalgebra $R\subset S=\prod_{i=1}^r \mathbb{K}[[t_i]]$. By a *cell* of λ, we mean an element in its domain of definition.

Lemma 2.5.1 ([BIV20, Remark 2.4]). *We have $\lambda(\alpha + e_i) - 1 \leq \lambda(\alpha) \leq \lambda(\alpha + e_i)$ where e_i is the i-th standard basis vector, i.e. from α to $\alpha + e_i$, λ either stays constant or increases by 1. Moreover, $\lambda(\alpha) = \lambda(\alpha + e_i)$ if and only if $\alpha[i]$ belongs to Σ. In particular, $\alpha \in \Sigma$ if and only if $\lambda(\alpha) = \lambda(\alpha + e_i)$ for all i.*

One implication of the above lemma is that if λ is a gap function with degree δ and $\lambda(\alpha) = \delta$ for some $\alpha \in \mathbb{Z}^r_{\geq 0}$, then by incrementing the components of α, the value of λ does not change.

Corollary 2.5.2. *A standard gap function λ is completely determined by its values on $\mathbb{Z}^r_{>0}$. In particular, for any $\alpha \in \mathbb{Z}^r_{\geq 0}$ with $\alpha_i = 0$, $\alpha \neq 0$, we have $\lambda(\alpha) = \lambda(\alpha + e_i) - 1$.*

Proof. Suppose λ is a standard gap function corresponding to $\mathbb{K}$-algebras $R \subset S = \prod_{i=1}^n \mathbb{K}[[t_i]]$. Being standard implies $t_i^0 \notin R$. Thus, if $\alpha = (\alpha_1, \ldots, \alpha_{i-1}, 0, \alpha_{i+1}, \ldots, \alpha_n) \neq 0$, we have

$$\begin{aligned}
\lambda(\alpha) &= \dim S/\left(R + \langle t_1^{\alpha_1}, \ldots, t_{i-1}^{\alpha_{i-1}}, t_i^0, t_{i+1}^{\alpha_{i+1}}, \ldots, t_n^{\alpha_n}\rangle\right) \\
&= \dim S/\left(R + \langle t_1^{\alpha_1}, \ldots, t_{i-1}^{\alpha_{i-1}}, t_i^1, t_{i+1}^{\alpha_{i+1}}, \ldots, t_n^{\alpha_n}\rangle\right) - 1 \\
&= \lambda(\alpha + e_i) - 1.
\end{aligned}$$

□

Example 2.5.3. Consider the gap function λ we computed in Example 2.4.6. The table at the end of the example shows the values of λ on $\mathbb{Z}^2_{>0}$. By Corollary 2.5.2, $\lambda(\alpha) = 0$ when α is on the horizontal or vertical axis.

Lemma 2.5.4 ([BIV20, Lemma 2.5]). *Fix $\gamma \in \mathbb{Z}_{\geq 0}$. Assume that $R \subset S$ is a subalgebra and let $\lambda = \lambda_R$. Suppose that for any $\alpha \in \mathbb{Z}^r_{\geq 0}$ satisfying $|\alpha| \leq 2\gamma + 2$, we have $\lambda(\alpha) \leq \gamma$. Then $\lambda(\alpha) \leq \gamma$ for all $\alpha \in \mathbb{Z}^r_{\geq 0}$, that is the degree of λ is at most γ.*

The significance of the above lemma is the following: suppose we are computing a gap function λ with degree δ. The above lemma tells us that λ attains the value δ at least once among all the points $\alpha \in \mathbb{Z}^r_{\geq 0}$ with $|\alpha| \leq 2(\delta - 1) + 2$. The next lemma imposes an even more strict condition on when δ appears for the first time as the value of λ.

Lemma 2.5.5 ([BIV20, Lemma 2.9]). *Let $R \subset S$ be a subalgebra and assume that $\lambda = \lambda_R$ is a standard gap function. Fix $\gamma \geq r - 1$. Consider any $\alpha \in \mathbb{Z}^r_{>0}$ with l coordinates α_i equal to one, such that $|\alpha| > 2\gamma + 2 - l$. Assume that for all $\alpha' \in \mathbb{Z}^r_{>0}$ with $\alpha' \neq \alpha$ and $(\alpha - \alpha')_i \geq 0$ for all i, we have $\lambda(\alpha') \leq \gamma$. Then $\lambda(\alpha) \leq \gamma$.*

The algorithm of Chapter 3 produces tables that are candidates for being gap functions. For each table λ in the output, to prove it is a gap function, the final step will be to produce the subalgebra $R \subset S$ such that $\lambda = \lambda_R$. The next lemma will help us produce the suitable R.

1.1:	**0** **1** 1 ...
1.2:	⋮ ⋮ 1 1 ... **1** 1 ...

Table 2.1: Gap functions with $\delta = 1$

Lemma 2.5.6 ([BIV20, Lemma 2.10])**.** *Assume that R is a complete subalgebra of S, that $\lambda = \lambda_R$ is a standard gap function, and that $\dim S/R$ is finite. Fix some $1 \leq i \leq r$. Consider $\alpha \in \mathbb{Z}^r_{>0}$ such that for any $\alpha' \in \mathbb{Z}^r_{>0}$ with $\alpha'_i = \alpha_i$ and $\alpha'_j \geq \alpha_j$ for all $j \neq i$, $\lambda(\alpha' + e_i) = \lambda(\alpha')$. Then for some unit $u \in \mathbb{K}[[t_i]]$, $ut_i^{\alpha_i} \in R$.*

The next lemma states that the number of gap functions of a given degree is finite and that we can store a gap function in a finite amount of memory in a computer.

Lemma 2.5.7 ([BIV20, Remark 5.1])**.** *For a fixed δ, there are only finitely many possible standard gap functions λ of degree δ coming from subalgebras of S. Moreover, λ is determined by its values on those $\alpha \in \mathbb{Z}^r_{>0}$ satisfying $|\alpha| \leq 2\delta$.*

Proof. By definition, a standard gap function of degree δ can have dimension at most $r = \delta + 1$. Fix δ and $r \leq \delta + 1$. Let λ be a standard r-dimensional gap function of degree δ coming from a subalgebra of $S = \prod_{i=1}^r \mathbb{K}[[t_i]]$. By Corollary 2.5.2, λ is determined by its values on $\mathbb{Z}^r_{>0}$. Now assume $\alpha \in \mathbb{Z}^r_{\geq 0}$ is such that $|\alpha| > 2\delta$. Define $\gamma = \max_{\alpha'} \lambda(\alpha')$ where the maximum is taken over those $\alpha' \in \mathbb{Z}^r_{\geq 0}$ such that $\alpha'_i \leq \alpha_i$ for every i and $\alpha' \neq \alpha$. We claim $\lambda(\alpha) = \gamma$. By Lemma 2.5.1, $\lambda(\alpha) \geq \gamma$. If $\gamma = \delta$, then we already have $\lambda(\alpha) = \gamma$. If $\gamma < \delta$, then $|\alpha| > 2\delta \geq 2\gamma + 2$, and by Lemma 2.5.5, we conclude $\lambda(\alpha) = \gamma$. In this way, by having the values of λ at the entries α with $|\alpha| \leq 2\delta$, we can inductively determine the values of λ everywhere. □

Lemma 2.5.8 (Upward propagation, [BIV20, Remark 5.2])**.** *Suppose λ is a standard gap function and $\alpha \in \mathbb{Z}^r_{\geq 0}$. Fix an integer $1 \leq i \leq r$. If $\lambda(\alpha) < \lambda(\alpha + e_i)$, then for $\alpha' \in \mathbb{Z}^r_{\geq 0}$ such that $\alpha'_j \geq \alpha_j$ for all j and $\alpha'_i = \alpha_i$, we have $\lambda(\alpha') < \lambda(\alpha' + e_i)$. We call such a behaviour upward propagation.*

By using the lemmas in this section, the authors in [BIV20] have classified the standard gap functions of degree at most 3. We record this result in the following proposition.

Proposition 2.5.9. *The positive standard gap functions λ for subalgebras R of S with $\delta(\lambda) \leq 3$ are, up to permutation of the r coordinates, those listed in Table 2.1, 2.2 and 2.3 .*

2.1.1:	**0**	**1**	**2**	...	**2.1.2:**	**0**	**1**	1	**2**	...
2.2.1:	⋮	⋮			**2.2.2:**	⋮	⋮			
	1	2	...			1	2	...		
	1	1	...			**1**	**2**	...		

Table 2.2: Gap functions with $\delta = 2$

2.6 Further properties of gap gunctions

In this section, we will state several properties of gap functions that do not appear in the literature. Since these results are new, we will provide proofs. All of these properties are about building new gap functions from existing ones by either projection from one gap function or assembling two gap functions.

Let R be a subalgebra of S as before and suppose λ_R is the corresponding gap function. We say λ_R is an r *dimensional* gap function if its domain of definition is $\mathbb{Z}_{\geq 0}^r$. The first operation on gap functions that we will discuss is projection, which produces a lower dimensional gap function.

Definition 2.6.1. Let $\lambda_R : \mathbb{Z}_{\geq 0}^r \to \mathbb{Z}_{\geq 0}$ be the gap function corresponding to $R \subset S$. Fix $1 \leq i \leq r$. The *projection of* λ_R *onto its* i*-th hyperplane* is the function

$$\begin{aligned} \lambda_{R,H_i} &: \mathbb{Z}_{\geq 0}^{r-1} \to \mathbb{Z}_{\geq 0} \\ (\alpha_1, \ldots, \alpha_{r-1}) &\mapsto \lambda_R(\alpha_1, \ldots, \alpha_{i-1}, 0, \alpha_i, \ldots, \alpha_{r-1}). \end{aligned}$$

Similarly, the *projection of* λ_R *onto its* i*-th axis* is the function

$$\begin{aligned} \lambda_{R,L_i} &: \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0} \\ &\alpha_1 \mapsto \lambda_R(0, \ldots, 0, \alpha_1, 0, \ldots, 0), \end{aligned}$$

where α_1 appears in i-th position. When R is known from context, we denote the projections simply by λ_{H_i} and λ_{L_i}.

Lemma 2.6.2. *Let* $\lambda : \mathbb{Z}_{\geq 0}^r \to \mathbb{Z}_{\geq 0}$ *be a standard gap function and fix* $1 \leq i \leq r$*. Then, the projections* λ_{H_i} *and* λ_{L_i} *are standard gap functions.*

Proof. Let λ be a standard gap function corresponding to a subalgebra $R \subset S = \prod_{j=1}^r \mathbb{K}[[t_j]]$. To show λ_{H_i} is a gap function, let $I = \langle t_i^0 \rangle$, $S' = S/I$ and $R' = R/I$. Then the gap function

3.1.1:	**0**	**1**	**2**	**3**	…	**3.1.2:**	**0**	**1**	**2**	2	**3**	…
3.1.3:	**0**	**1**	**2**	2	2	**3**	…	**3.1.4:**	**0**	**1**	1	**2**	2	**3**	…

3.2.1:	⋮	⋮	⋮			**3.2.2:**	⋮	⋮	⋮	⋮	
	1	2	3	…			1	2	2	3	…
	1	2	3	…			1	2	2	3	…
	1	**2**	**3**	…			**1**	**2**	2	**3**	…
3.2.3:	⋮	⋮	⋮			**3.2.4:**	⋮	⋮	⋮		
	2	3	3	…			1	2	3	…	
	2	**3**	3	…			1	2	**3**	…	
	1	**2**	2	…			**1**	**2**	2	…	
3.2.5:	⋮	⋮	⋮	⋮		**3.2.6:**	⋮	⋮	⋮		
	1	2	2	3	…		1	2	**3**	…	
	1	2	2	**3**	…		1	**2**	2	…	
	1	**2**	2	2	…		**1**	1	1	…	

3.3.1:	$\lambda(1,1,1)=2,$	$\lambda(1,1,2)=3$
3.3.2:	$\lambda(1,1,1)=2,$	$\lambda(1,2,2)=3$
3.3.3:	$\lambda(1,1,1)=2,$	$\lambda(2,2,2)=3$
3.4:	$\lambda(1,1,1,1)=3$	

Table 2.3: Gap functions with $\delta = 3$

of the subalgebra $R' \subset S'$ is the map $\lambda_{R'} : \mathbb{Z}_{\geq 0}^{r-1} \to \mathbb{Z}_{\geq 0}$, where

$$\begin{aligned}
\lambda_{R'}(\alpha_1,\ldots,\alpha_{r-1}) &= \dim \frac{S/I}{R/I + \langle t_1^{\alpha_1},\ldots,t_{i-1}^{\alpha_{i-1}},t_{i+1}^{\alpha_i},\ldots,t_r^{\alpha_{r-1}}\rangle} \\
&= \dim \frac{S}{R + \langle t_1^{\alpha_1},\ldots,t_{i-1}^{\alpha_{i-1}},t_i^0,t_{i+1}^{\alpha_i},\ldots,t_r^{\alpha_{r-1}}\rangle} \\
&= \lambda(\alpha_1,\ldots,\alpha_{i-1},0,\alpha_i,\ldots,\alpha_{r-1}) \\
&= \lambda_{H_i}(\alpha_1,\ldots,\alpha_{r-1}).
\end{aligned}$$

Because λ is standard, $t_j^0 \notin R$ for all $1 \leq j \leq r$, and the same is true for R'. Therefore $\lambda_{R'} = \lambda_{H_i}$ is a standard gap function. By using the same argument with the ideal $I = \langle t_1^0,\ldots,t_{i-1}^0,t_{i+1}^0,\ldots,t_r^0\rangle$, we can prove that λ_{L_i} is also a standard gap function. □

This lemma will be crucial in our algorithm for producing gap functions because it allows us to build an r dimensional gap function by assembling an $(r-1)$-dimensional and a 1-dimensional gap functions. The next lemma tells us what the degrees of the smaller gap functions in the assembly can be.

Lemma 2.6.3. *Let $\lambda : \mathbb{Z}^r_{\geq 0} \to \mathbb{Z}_{\geq 0}$ be a standard gap function with projections λ_{H_i} and λ_{L_i}, where $1 \leq i \leq r$ is fixed. Then,*

$$\delta(\lambda) \geq \delta(\lambda_{H_i}) + \delta(\lambda_{L_i}) + 1.$$

Proof. We prove the lemma for $i = r$. The proof for other values of i are similar. Suppose λ_{H_i} takes the value $\delta(\lambda_{H_i})$ at $(\alpha_1, \ldots, \alpha_{r-1})$. Then

$$\lambda(\alpha_1, \ldots, \alpha_{r-1}, 0) = \delta(\lambda_{H_i}).$$

By upward propagation (Lemma 2.5.8) and

$$\lambda(1, 1, \ldots, 1, 0) = r - 2,$$
$$\lambda(1, 1, \ldots, 1, 1) = r - 1,$$

we have

$$\lambda(\alpha_1, \ldots, \alpha_{r-1}, 1) = \delta(\lambda_{H_i}) + 1. \tag{2.1}$$

Consider the path in $\mathbb{Z}^r_{\geq 0}$ given by

$$(0, \ldots, 0, 1) \to (0, \ldots, 0, 2) \to \cdots \to (0, \ldots, 0, k),$$

where k is such that $\lambda(0, \ldots, 0, k) = \lambda_{L_i}(k) = \delta(\lambda_{L_i})$. By applying upward propagation to Equation (2.1) and

$$\lambda(0, \ldots, 0, 1) = \lambda_{L_i}(1) = 0,$$

we have

$$\lambda(\alpha_1, \ldots, \alpha_{r-1}, k) \geq (\delta(\lambda_{H_i}) + 1) + \delta(\lambda_{L_i}).$$

This means $\delta(\lambda) \geq \delta(\lambda_{H_i}) + \delta(\lambda_{L_i}) + 1$. □

We will see in Proposition 2.6.6 that in the above lemma, the situation where $\delta(\lambda) = \delta(\lambda_{H_i}) + \delta(\lambda_{L_i}) + 1$ is very special.

Notation 2.6.4. For a power series $f(t) \in \mathbb{K}[[t]]$, we call the coefficient of t^0 the *constant term* of f and denote it by $f(0)$. If $g = (f_1(t_1), f_2(t_2), \ldots, f_r(t_r)) \in \prod_{i=1}^r \mathbb{K}[[t_i]]$ is an r-tuple of power series, we define

$$g(0) = (f_1(0), f_2(0), \ldots, f_r(0)).$$

Lemma 2.6.5. *Let $\lambda : \mathbb{Z}^r_{\geq 0} \to \mathbb{Z}_{\geq 0}$ be a standard gap function corresponding to a subalgebra*

$$R = \mathbb{K}[[f_1, \ldots, f_\ell]] \subset S = \prod_{i=1}^{r} \mathbb{K}[[t_i]],$$

where $f_i = (f_{i,1}(t_1), \ldots, f_{i,r}(t_r)) \in S$. Then for fixed i, the constant terms of $f_{i,j}$ are equal. This shows we may assume the constant terms of $f_{i,j}$ are all zero.

Proof. By assumption

$$\begin{aligned}\lambda(1,1,\ldots,1) &= \dim \frac{S}{R + \langle t_1, \ldots, t_r \rangle} \\ &= \dim \frac{\mathbb{K} \times \ldots \times \mathbb{K}}{\operatorname{span}\{(1,1,\ldots,1), f_1(0), \ldots, f_\ell(0)\}} \\ &= r - 1,\end{aligned}$$

where on the first line, $\mathbb{K}$ is repeated r-times. This shows that $f_i(0) \in \operatorname{span}\{(1,1,\ldots,1)\}$. □

Proposition 2.6.6. *Assume $\lambda_H : \mathbb{Z}_{\geq 0}^{r-1} \to \mathbb{Z}_{\geq 0}$ and $\lambda_L : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ are two standard gap functions associated to subalgebras*

$$\begin{aligned}R_1 = \mathbb{K}[[f_1, \ldots, f_m]] \subset S_1 = \prod_{i=1}^{r-1} \mathbb{K}[[t_i]], \\ R_2 = \mathbb{K}[[g_1, \ldots, g_n]] \subset S_2 = \mathbb{K}[[t_r]],\end{aligned}$$

respectively. If $\delta = \delta(\lambda_H) + \delta(\lambda_L) + 1$, then

1. *There exists a unique standard gap function $\lambda : \mathbb{Z}_{\geq 0}^{r} \to \mathbb{Z}_{\geq 0}$ with degree δ such that its projections λ_{H_n} and λ_{L_n} are equal to λ_H and λ_L, respectively.*

2. *Assume the subalgebras giving the gap functions λ_H and λ_L are unique up to automorphisms of S_1 and S_2 respectively. Then λ in part 1 is the gap function of the unique subalgebra*

$$\mathbb{K}[[f_1, \ldots, f_m, g_1, \ldots, g_n]] \subset S = \prod_{i=1}^{r} \mathbb{K}[[t_i]],$$

up to automorphisms of S.

Proof. Statement 1: To show the existence, let λ be the gap function of

$$R \coloneqq \mathbb{K}[[f_1, \ldots, f_m, g_1, \ldots, g_n]] \subset S.$$

From the definition of R and by using the proof of Lemma 2.6.2 we have $\lambda_{H_n} = \lambda_H$ and $\lambda_{L_n} = \lambda_L$. To show λ has degree δ, we prove that for $\alpha_1, \ldots, \alpha_r \geq 1$

$$\lambda(\alpha_1, \ldots, \alpha_r) = \lambda_H(\alpha_1, \ldots, \alpha_{r-1}) + \lambda_L(\alpha_r) + 1,$$

which is the same as showing

$$\dim \frac{S}{R + \langle t_1^{\alpha_1}, \ldots, t_r^{\alpha_r} \rangle} = \dim \frac{S_1}{R_1 + \langle t_1^{\alpha_1}, \ldots, t_{r-1}^{\alpha_{r-1}} \rangle} + \dim \frac{S_2}{R_2 + \langle t_r^{\alpha_r} \rangle} + 1.$$

It is enough to show that there is a vector space isomorphism

$$\frac{S}{\mathbb{K}[[f_1,\dots,f_m,g_1,\dots,g_n]]+\langle t_1^{\alpha_1},\dots,t_r^{\alpha_r}\rangle}\cong\frac{S_1}{\mathbb{K}[[f_1,\dots,f_m]]+\langle t_1^{\alpha_1},\dots,t_{r-1}^{\alpha_{r-1}}\rangle}\times\frac{S_2}{\mathbb{K}[[g_1,\dots,g_n]]+\langle t_r^{\alpha_r}\rangle}\times\mathbb{K}.$$

So, we need to define a linear map:

$$\phi: S\to\frac{S_1}{\mathbb{K}[[f_i]]+\langle t_1^{\alpha_1},\dots,t_{r-1}^{\alpha_{r-1}}\rangle}\times\frac{S_2}{\mathbb{K}[[g_j]]+\langle t_r^{\alpha_r}\rangle}\times\mathbb{K},$$

such that $\ker\phi=\mathbb{K}[[f_1,\dots,f_m,g_1,\dots,g_n]]+\langle t_1^{\alpha_1},\dots,t_r^{\alpha_r}\rangle$.

For $(h_1(t_1),h_2(t_2),\dots,h_r(t_r))\in S$, where $h_i(t_i)\in\mathbb{K}[[t_i]]$, define ϕ by

$$(h_1(t_1),\dots,h_r(t_r))\mapsto\left(\overline{(h_1(t_1),\dots,h_{r-1}(t_{r-1}))},\overline{h_r(t_r)},h_1(0)-h_r(0)\right).$$

We have to show that it is surjective and the kernel is as above. First, we need to show the surjectivity: we take an arbitrary element in the right hand side as

$$\left(\overline{(h_1(t_1),\dots,h_{r-1}(t_{r-1}))},\overline{h_r(t_r)},C\right),$$

where $C\in\mathbb{K}$. After adding $(c,c,\dots,c)\in\mathbb{K}[[f_i]]$ for some suitable $c\in\mathbb{K}$ to $(h_1,\dots,h_{r-1})$, we can assume that $h_1(t_1)$ does not have a constant term. Similarly we may assume $h_r(t_r)$ does not have a constant term. Now, we calculate

$$\begin{aligned}\phi(h_1+C,h_2+C,\dots,h_{r-1}+C,h_r)&=\left(\overline{(h_1+C,\dots,h_{r-1}+C)},\overline{h_r},h_1(0)+C-h_r(0)\right)\\&=\left(\overline{(h_1,\dots,h_{r-1})},\overline{h_r},C\right),\end{aligned}$$

which shows that ϕ is surjective. Now, it is left to show that

$$\ker\phi=\mathbb{K}[[f_1,\dots,f_m,g_1,\dots,g_n]]+\langle t_1^{\alpha_1},\dots,t_r^{\alpha_r}\rangle.$$

First, we show that $\mathbb{K}[[f_1,\dots,f_m,g_1,\dots,g_n]]+\langle t_1^{\alpha_1},\dots,t_r^{\alpha_r}\rangle\subset\ker\phi$. We take an element $(H_1(t_1),\dots,H_r(t_r))+(H_1'(t_1),\dots,H_r'(t_r))\in\mathbb{K}[[f_1,\dots,f_m,g_1,\dots,g_n]]+\langle t_1^{\alpha_1},\dots,t_r^{\alpha_r}\rangle$, where $H_i(t_i),H_i'(t_i)\in\mathbb{K}[[t_i]]$, such that $(H_1,\dots,H_r)\in\mathbb{K}[[f_1,\dots,f_m,g_1,\dots,g_n]]$ and $(H_1',\dots,H_r')\in\langle t_1^{\alpha_1},\dots,t_r^{\alpha_r}\rangle$. Our goal is to show $\phi(H_1+H_1',\dots,H_r+H_r')=0$.

Because λ_H and λ_L are standard, we may assume the generators f_i,g_j do not have a constant term (see Lemma 2.6.5). Therefore, $H_1,\dots,H_r$ have the same constant term. Similarly since $\alpha_1,\dots,\alpha_r\geq 1$, we have that $H_1',\dots,H_r'$ do not have a constant term.

Therefore,

$$\begin{aligned}
&\phi(H_1 + H_1', \ldots, H_r + H_r') \\
&= \Big((\overline{H_1 + H_1', H_2 + H_2', \ldots, H_{r-1} + H_{r-1}'}), \overline{H_r + H_r'}, H_1(0) + H_1'(0) - H_r(0) - H_r'(0)\Big) \\
&= ((\overline{0, \ldots, 0}), \overline{0}, 0) = 0.
\end{aligned}$$

Now, in order to show that $\ker \phi \subset \mathbb{K}[[f_1, \ldots, f_m, g_1, \ldots, g_n]] + \langle t_1^{\alpha_1}, \ldots, t_r^{\alpha_r} \rangle$, we take an element $(H_1(t_1), \ldots, H_r(t_r))$ in $\ker \phi$ where $h_i(t_i) \in \mathbb{K}[[t_i]]$:

$$\phi(H_1(t_1), \ldots, H_r(t_r)) = 0 \Rightarrow \Big((\overline{H_1, \ldots, H_{r-1}}), \overline{H_r}, H_1(0) - H_r(0)\Big) = 0.$$

Thus, $(H_1, \ldots, H_{r-1}) \in \mathbb{K}[[f_1, \ldots, f_m]] + \langle t_1^{\alpha_1}, \ldots, t_r^{\alpha_r} \rangle$, $H_r \in \mathbb{K}[[g_1, \ldots, g_n]] + \langle t_r^{\alpha_r} \rangle$ and the H_i's have the same constant ($H_1, \ldots, H_{r-1}$ have the same constant because λ_H is standard and the previous equation gives $H_1(0) = H_r(0)$). Then,

$$\begin{aligned}
(H_1(t_1), \ldots, H_{r-1}(t_{r-1}), H_1(0)) &\in \mathbb{K}[[f_i, g_j]], \\
(0, \ldots 0, H_r(t_r) - H_r(0)) &\in \mathbb{K}[[f_i, g_j]].
\end{aligned}$$

Adding the two gives $(H_1, \ldots, H_r) \in \mathbb{K}[[f_1, \ldots, f_m, g_1, \ldots, g_n]] + \langle t_1^{\alpha_1}, \ldots, t_r^{\alpha_r} \rangle$. By the first homomorphism theorem, the claim is proved.

The uniqueness of λ follows from upward propagation (Lemma 2.5.8). Indeed, to get

$$\delta(\lambda) = \delta(\lambda_H) + \delta(\lambda_L) + 1,$$

the gap function λ increases value from $\alpha \in \mathbb{Z}_{\geq 0}^r$ to $\alpha + e_i$ exactly according to whether λ_H or λ_L increases from the projection of α to projection of $\alpha + e_i$. Any extra increases in λ will force $\delta(\lambda)$ to be larger than $\delta(\lambda_H) + \delta(\lambda_L) + 1$.

Statement 2: Let $R' \coloneqq \mathbb{K}[[h_1, \ldots, h_\ell]] \subset S$ be a subalgebra with gap function λ. Let p be the projection of S onto the first $n-1$ components and let p_r be the projection of S onto the last component

$$\begin{aligned}
p &: S \to \prod_{i=1}^{r-1} \mathbb{K}[[t_i]], \\
p_r &: S \to \prod \mathbb{K}[[t_r]].
\end{aligned}$$

Consider the subalgebra $\tilde{R} \subset S$ generated by all $p(h_i)$ and $p_r(h_i)$ (for $1 \leq i \leq \ell$). Denote the gap function of $\tilde{R}$ by $\tilde{\lambda}$. By the proof of Lemma 2.6.2, $\tilde{\lambda}_{H_n} = \lambda_{H_n}$ is the gap function of the subalgebra of S_1 generated by all $p(h_i)$ and $\tilde{\lambda}_{L_n} = \lambda_{L_n}$ is the gap function of the subalgebra of S_2 generated by all $p_r(h_i)$. By the proof of statement 1, we conclude $\tilde{\lambda} = \lambda$. We now prove $R' = \tilde{R}$. It is obvious that $R' \subset \tilde{R}$. If we look at the surjective linear map of

vector spaces $\psi : S/R' \to S/\tilde{R}$, because

$$\dim S/R' = \delta = \dim S/\tilde{R},$$

we conclude that ψ is an isomorphism, i.e. $R' = \tilde{R}$.

Finally, because $\tilde{\lambda}_{H_n} = \lambda_H$ is the gap function of the subalgebra of S_1 generated by all $p(h_i)$, by hypothesis there is an automorphism of S_1 that restricts to an isomorphism

$$\mathbb{K}[[p(h_1), \ldots, p(h_\ell)]] \cong \mathbb{K}[[f_1, \ldots, f_m]].$$

Similarly, there is an automorphism of S_2 that restricts to an isomorphism

$$\mathbb{K}[[p_r(h_1), \ldots, p_r(h_\ell)]] \cong \mathbb{K}[[g_1, \ldots, g_n]].$$

The above automorphisms of S_1 and S_2 combine to an automorphism of S that restricts to an isomorphism

$$R' = \mathbb{K}[[p(h_1), \ldots, p(h_\ell), p_r(h_1), \ldots, p_r(h_\ell)]] \cong \mathbb{K}[[f_1, \ldots, f_m, g_1, \ldots, g_n]].$$

□

Definition 2.6.7. We call the unique λ in Proposition 2.6.6, the *product* of the gap functions λ_H and λ_L.

The fact that the product of two gap functions is again a gap function will help us in the next chapter to immediately identify some of the outputs of the algorithm as gap functions.

2.7 Geometry of gap functions

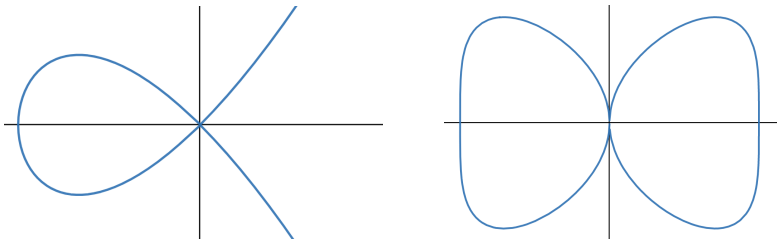
Let P be a singular point on a curve C. Assume $\lambda : \mathbb{Z}^r_{\geq 0} \to \mathbb{Z}_{\geq 0}$ is the gap function of the subalgebra

$$\hat{\mathcal{O}}_{C,P} \hookrightarrow \prod_{i=1}^{r} \mathbb{K}[[t_i]],$$

where r is the number of branches at P. For a fixed $1 \leq i \leq r$, the projection λ_{L_i} of λ onto its i-th component is the gap function describing the i-th branch of the singularity at P. Moreover, λ describes how the r branches intersect at P. By using Lemma 2.6.3 r-times, we have

$$\delta(\lambda) \geq \sum_{i=1}^{r} \delta(\lambda_{L_i}) + (r-1). \tag{2.2}$$

Here equality occurs when the branches intersect transversely, i.e. the intersection of the tangent space of one branch with the tangent space of the remaining $r-1$ branches is zero. If this intersection is not zero, then $\delta(\lambda)$ will be larger and equality does not hold.



(a) The point (0,0) on the curve $y^2 - x^2(x+1) = 0$ is a simple node. The two branches intersect transversely.

(b) The point (0,0) on the curve $x^2 - x^4 - y^4 = 0$ is a tacnode. The two branches intersect non-transversely.

Figure 2.4: The singularities in Example 2.7.1

Example 2.7.1. Let λ be the gap function **1.2** in Table 2.1. Here, the projections $\lambda_{L_1}, \lambda_{L_2} : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ are the zero function. This means that λ describes a singularity with two smooth branches (see Theorem 2.2.4) which intersect transversely. An example of this situation is a simple node (see Figure 2.4a).

As another example, let λ' be the gap function **2.2.1** in Table 2.2. Again there are two smooth branches but they do not intersect transversely since for λ', equality does not hold in (2.2). The two branches have the same tangent direction at the singular point (see Figure 2.4b).

Now we can interpret the product of two gap functions geometrically. Suppose λ_L, λ_H are gap functions corresponding to singularities with one branch and $r-1$ branches, respectively. Then the product of λ_L and λ_H describes the unique singularity type where the single branch corresponding to λ_L intersects transversely the singular point with $r-1$ branches coming from λ_H.

Chapter 3

Algorithm for computing gap functions

In this chapter, we introduce an algorithm that computes tables that are candidates for being standard gap functions. From here on, we refer to standard gap functions simply as gap functions. As input, the algorithm takes an integer δ and a list of gap functions with degree less than δ. The output is a list of tables Λ such that the set of all gap functions with degree δ is a subset of Λ. A key challenge is to make Λ as small as possible. We will use the properties of gap functions in Sections 2.4, 2.5 and 2.6 to eliminate as many redundant and non-gap tables as possible. Ideally we want the output to be equal to the set of all gap functions with degree δ up to permutations of the components. We will see that for $\delta \leq 4$, the algorithm is correct.

3.1 Data structure for representing a gap function

We saw in Lemma 2.5.7 that a gap function can be represented as a finite table of values. More specifically, we can store a gap function $\lambda : \mathbb{Z}^r_{\geq 0} \to \mathbb{Z}_{\geq 0}$ as a finite r-dimensional array of non-negative integers. Moreover, there is a *minimal* array that represents λ. Before we give the exact statement in Lemma 3.1.2, we need to define a partial ordering on $\mathbb{Z}^r_{\geq 0}$.

Definition 3.1.1. Let $r \geq 1$. We define the partial ordering $\preccurlyeq$ on $\mathbb{Z}^r_{\geq 0}$ as follows: For $\alpha, \alpha' \in \mathbb{Z}^r_{\geq 0}$, we write $\alpha \preccurlyeq \alpha'$ if for every $1 \leq i \leq r$, $\alpha_i \leq \alpha'_i$.

Lemma 3.1.2. *Let $\lambda : \mathbb{Z}^r_{\geq 0} \to \mathbb{Z}_{\geq 0}$ be standard gap function with degree δ. Then, the set $A = \{\alpha \in \mathbb{Z}^r_{\geq 0} : \lambda(\alpha) = \delta\}$ has an element $\tilde{\alpha}$ such that $\tilde{\alpha} \preccurlyeq \alpha$ for every $\alpha \in A$.*

Proof. By definition, A is non-empty, and since for every $\alpha' \in \mathbb{Z}^r_{\geq 0}$, the set $\{\alpha \in \mathbb{Z}^r_{\geq 0} : \alpha \preccurlyeq \alpha'\}$ is finite, A has a minimal element with respect to $\preccurlyeq$. We claim this minimal element is the desired $\tilde{\alpha}$. If $\alpha \in A$, by minimality of $\tilde{\alpha}$, either $\tilde{\alpha} \preccurlyeq \alpha$ or $\tilde{\alpha}, \alpha$ are not comparable. We claim the latter cannot happen. Define $\hat{\alpha} \in \mathbb{Z}^r_{\geq 0}$ by $\hat{\alpha}_i = \min\{\alpha_i, \tilde{\alpha}_i\}$ for every $1 \leq i \leq r$. By definition, $\hat{\alpha} \prec \tilde{\alpha}$. Thus, $\lambda(\hat{\alpha}) < \delta$. Build a path from $\hat{\alpha}$ to α by using the standard basis

vectors as steps. Next, use the same sequence of steps to build a path starting from $\tilde{\alpha}$ and ending in some $\beta \in \mathbb{Z}^r_{\geq 0}$. Now, upward propagation (Lemma 2.5.8) implies $\lambda(\beta) > \lambda(\tilde{\alpha}) = \delta$, a contradiction. □

Definition 3.1.3. For a standard gap function $\lambda : \mathbb{Z}^r_{\geq 0} \to \mathbb{Z}_{\geq 0}$, we call $\tilde{\alpha}$ in Lemma 3.1.2, the *final index* or *size of* λ.

Definition 3.1.4. We define `GapFunction` to be the data structure that can store the data of a gap function. It is a finite r-dimensional array of non-negative integers whose indexing starts from zero. If λ is a variable of type `GapFunction`, the following table shows our convention for accessing the data stored in λ.

λ.size	largest index of the array with respect to $\preccurlyeq$
λ.dim	the dimension of the array, i.e. r
$\lambda(\alpha)$	the value of the array at index $\alpha \in \mathbb{Z}^r_{\geq 0}$
λ.degree	the value at the largest index, i.e. $\lambda(\lambda.\text{size})$

Remark 3.1.5. We distinguish between the terms "`GapFunction`" and "gap function". The latter is a function $\mathbb{Z}^r_{\geq 0} \to \mathbb{Z}_{\geq 0}$ as in Definition 2.4.1. The former is an array of non-negative integers that may or may not contain the valid data of a gap function. When we say a `GapFunction` is a gap function, we mean the array contains the data of an actual gap function (coming from $\mathbb{K}$-algebras $R \subset S$).

3.2 Description of algorithm

The input and output of the algorithm we will present for computing potential gap functions is as follows:

Input	an integer $\delta > 0$, a list of all gap functions with degree less than δ
Output	a set L of `GapFunctions` with dimension at most $\delta + 1$ such that the set of all gap functions with degree δ is a subset of L

Recall that by definition, the dimension of a standard gap function can be at most one more than the degree. We present the algorithm in several procedures. The first procedure of the algorithm is called `fillGapFunction` which takes as input a partially filled `GapFunction` and tries to fill the rest of the cells according to the properties satisfied by a gap function. This procedure starts from a given cell, and fills the next cell, then the next until it gets to the last index. To understand what we mean by the next cell, let us first describe the procedure *increment*.

3.2.1 Procedure `increment`

This procedure (pseudocode on page 31) takes a starting index $\alpha^{(s)}$ and a final index $\alpha^{(f)}$. It then produces the index α right after $\alpha^{(s)}$ by incrementing the first component of $\alpha^{(s)}$. If this incremented first component is not more than the first component of $\alpha^{(f)}$, the new index is returned (along with the boolean value true, which means the incrementing operation is successful). Otherwise, it sets the first component to 1 and increments the second component by 1, then compares it with the second component of $\alpha^{(f)}$. The procedure continues in this way. If $\alpha^{(s)} \prec \alpha^{(f)}$, then there always exists a next index which will be the output of the procedure along the boolean value true. If $\alpha^{(f)} \preccurlyeq \alpha^{(s)}$, then there is no next index and the procedure returns $\alpha^{(f)}$ along with the boolean value false.

Procedure 1: increment

input : $\alpha^{(s)} \in \mathbb{Z}^r_{>0}$(the current index), $\alpha^{(f)} \in \mathbb{Z}^r_{>0}$(the final index)
output: true or false, $\alpha^{(n)} \in \mathbb{Z}^r_{>0}$(the next index)

```
1  α^(n) ← α^(s);
2  r ← the number of components of α^(s);
3  α_1^(n) ← α_1^(n) + 1;
4  for i = 1 to r do
5  |  if α_i^(n) ≤ α_i^(f) then
6  |  |  return (true,α^(n));
7  |  end
8  |  else if α_i^(n) > α_i^(f) and i < r then
9  |  |  α_{i+1}^(n) ← α_{i+1}^(n) + 1;
10 |  |  α_i^(n) ← 1;
11 |  else
12 |  |  return (false, α^(f));
13 |  end
14 end
```

3.2.2 Procedure `main`

The procedure `main` (pseudocode on page 33) is the entry point into our algorithm. The input to main are the degree (δ), dimension (r) and a list (called `gapFunList`) of all gap functions with degree less than δ. The output of `main` is a set of `GapFunction`s that are candidates for being gap functions with degree δ and dimension r.

First we describe what it does and then explain the pseudocode step by step: When we compute a `GapFunction` λ, it is the procedure `fillGapFunction` that fills in the majority of the entries of λ. The procedure `fillGapFunction` accepts a partially filled `GapFunction` (whose entries on the coordinate hyperplanes are all filled in) and then completely fills the `GapFunction`. Before we can pass a `GapFunction` λ to `fillGapFunction`, the entries of λ

on the coordinate hyperplanes (i.e. those entries $\alpha \in \mathbb{Z}^r_{\geq 0}$ with at least one zero component), needs to be filled. If α is a one dimensional array (i.e. λ.dim$= 1$), this is already done since in this case $\lambda(0) = \lambda(1) = 0$, and `fillGapFunction` can start filling from $\alpha_{\text{start}} = 1$ (recall that α_{start}, the starting entry for `fillGapFunction` must already be filled in). However, for higher dimensional gap functions, we need to prepare the table λ, i.e. fill all the entries on the coordinate hyperplanes of λ and then by setting $\lambda(1, 1, \ldots, 1) = r - 1$ (since λ is standard), we can pass λ to `fillGapFunction` along with $\alpha_{\text{start}} = (1, 1, \ldots, 1)$. The procedure `main` does this preparation.

In Lines 1-10 we handle the 1-dimensional gap functions. First we check if the input δ is zero. If so there is nothing to compute since there is only one gap function with degree zero, namely the 1-dimensional zero function. If $\delta > 0$ then we create a new 1-dimensional array with size 2δ (recall that the arrays are zero indexed and the size is the index of the last entry and not the number of entries), set the entries $\lambda(0), \lambda(1)$ to zero and pass this partially filled λ to `fillGapFunction`. Starting from Line 11, we handle the r-dimensional case when $r > 1$. For computing gap function candidates with dimension $r > 1$, the procedure `main` assembles a one-dimensional `GapFunction` (named `axis`) and an $(r-1)$-dimensional `GapFunction` (named `hyperplane`) by using another procedure called `assembleGapFunction` which in turn calls `fillGapFunction` to produce the ultimate result. We will later give the details of the procedure `assembleGapFunction`. In Lines 12-14 we create two lists. The lists `axisList` and `hyperplaneList` are populated with gap functions with dimension 1 and $r - 1$ respectively. These 1- and $(r-1)$-dimesnioal gap functions all have degree less than δ and are available in the input `gapFunList`. For each pair (`axis`, `hyperplane`) where `axis` is in `axisList` and `hyperplane` is in `hyperplaneList` if the condition

$$\texttt{axis}.\text{degree} + \texttt{hyperplane}.\text{degree} \leq \delta - 1.$$

is satisfied, we call the procedure `assebmleGapFunction` (in Line 18) to assemble the pair into an r-dimensional `GapFunction`. The above condition comes from Lemma 2.6.3. By iterating through all 1-dimensional and $(r-1)$-dimensional gap functions provided in `gapFunList` we practically build all `GapFunction`s that are candidates to be gap functions. Finally in Lines 19-23 we check to see the assembled gap functions are not repetitive (symmetry check).

Procedure 2: main

input : δ (an integer, the degree), r (an integer, the dimension), gapFunList (a list of all gap functions with degree less than δ)

output: Prints a set of GapFunctions that are candidates for being gap functions with degree δ and dimension r. Also, adds these GapFunctions to gapFunList.

```
1  if r = 1 then
2      if δ = 0 then
           /* if δ = 0, there is only one gap function which is the zero
              map ℤ_{≥0} → ℤ_{≥0}                                          */
3          print 0 and add to gapFunList;
4      end
       /* If δ > 0, we immediately use fillGapFunction                   */
5      s_x ← 2(δ − 1) + 2;
6      λ ← a new GapFunction with dimension 1 and size s_x;
7      λ(0) ← 0;
8      λ(1) ← 0;
9      fillGapFunction(λ, δ, α_start = 1, gapFunList)
10 end
11 if r ≥ 2 then
12     define lists axisList and hyperplaneList;
13     add all GapFunctions in gapFunList with dimension 1 to axisList;
14     add all GapFunctions in gapFunList with dimension r − 1 to hyperplaneList;
15     for axis in axisList do
16         for hyperplane in hyperplaneList do
17             if axis.degree+hyperplane.degree≤ δ − 1 then
                   /* assembled is a list of GapFunctions                 */
18                 assembled← assembleGapFunction(axis,hyperplane,δ);
19                 for gapFun in assembled do
20                     if no permutation of gapFun is in gapFunList then
21                         print gapFun and add it to gapFunList;
22                     end
23                 end
24             end
25         end
26     end
27 end
```

3.2.3 Procedure `assembleGapFunction`

The input to `assembleGapFunction` (pseudocode on page 35) is a pair: a 1-dimensional `GapFunction` named `axisGap` and an $(r-1)$-dimensional `GapFunction` named `hyperplaneGap`. The output is a list of `GapFunctions` λ whose projections λ_{L_1} and λ_{H_1} are exactly `axisGap` and `hyperplaneGap` respectively.

In Lines 3-13 we handle the 2-dimensional base case. The `if` in Line 6 verifies that Lemma 2.6.3 is satisfied for projections. If so, then a new 2-dimensional array λ with size $(2(\delta-1)+1, 2(\delta-1)+1)$ is created in Line 7. In Lines 8 and 9 we populate the axes of λ with entries in the 1-dimensional gap functions `axisGap` and `hyperplaneGap`. In Line 10 we set $\lambda(1,1)=1$ since λ is a standard gap function and then pass λ to `fillGapFunction` to be filled.

Starting from Line 14 we handle the r-dimensional case when $r>2$. The purpose is to build an r-dimensional `GapFunction` λ by assembling `axisGap` and `hyperplaneGap`. Line 15 creates the list `hListArray`. The i-th entry of this array will later be populated by those $(r-1)$-dimensional `GapFunctionss` that could be the projection of the desired λ onto its i-th hyperplane (i.e. λ_{H_i}). At this stage we only know the 1st entry in `hListArray`: It is a list containing only the `GapFunction hyperplaneGap` (Line 16).

In each iteration of the `for` loop in Lines 17-27 we use recursion to populate a list `assembled` of `GapFunctions` that will be the i-th entry in `hListArray`. Each `GapFunction` in `assembled` has dimension $r-1$ and the goal is to assemble it from `axisGap` (with dimension 1) and the projection of `hyperplaneGap` onto its $(i-1)$-th hyperplane which is an $(r-2)$-dimensional `GapFunction` (defined in Line 18). To assemble these two, we need to know the degree of the resulting $(r-1)$-dimensional `GapFunction`. In fact we do not exactly know what the degree is but we have an upper bound for the degree from Lemma 2.6.3. This upper bound is Δ in Line 19. The `for` loop in Lines 20-23 does the assemble for every degree in $0,\ldots,\Delta$ using recursion. The resulting `GapFunctions` are all added to the i-th list in `hListArray`. If nothing is added to the i-th list in `hListArray` (Line 24), then it is not possible to assemble `axisGap` and `hyperplaneGap` and we return an empty list.

Assuming that every entry of `hListArray` is a non-empty list, the `for` loop in Line 28 is executed. Here for every n-tuple $(\lambda_{H_1},\ldots,\lambda_{H_n})$ of $(r-1)$-dimensional `GapFunctions` where λ_{H_i} is in the list at the i-th entry of `hListArray`, we test whether or not $\lambda_{H_1},\ldots,\lambda_{H_n}$ are compatible (in Line 29). By compatible we mean that every pair $\lambda_{H_i},\lambda_{H_j}$ when considered as coordinate hyperplanes of λ, should have the same entries on the intersection of these two coordinate hyperplanes. The procedure tests all combinations, and for every compatible combination $(\lambda_{H_1},\ldots,\lambda_{H_r})$, it places the entries of λ_{H_i} as the entries of the i-th coordinate hyperplane of λ. We also set $\lambda(1,\ldots,1)=r-1$ because we want λ to be standard. The `GapFunction` λ is now ready to be passed to `fillGapFunction`.

Procedure 3: assembleGapFunction

input : axisGap (a GapFunction of dimension 1), hyperplaneGap (a GapFunction), δ (an integer)
output: a list of GapFunctions λ with degree δ such that the projections $\lambda_{L_1}, \lambda_{H_1}$ are exactly axisGap and hyperplaneGap (see Definition 2.6.1) for projections.

```
/* this procedure assumes axisGap has dimension 1                          */
r ← axisGap.dim+hyperplaneGap.dim;
gapFunList← an empty list of GapFunctions;
if r = 2 then
    L_x ← 2(δ − 1) + 1;
    L_y ← 2(δ − 1) + 1;
    if axisGap.degree+hyperplaneGap.degree≤ δ − 1 then
        λ ← an empty GapFunction with size (L_x, L_y);
        /* when r = 2, hyperplaneGap has dimension 1                       */
        /* copy the entries of axisGap to the entries on the first axis of λ */
        λ(i, 0) ← axisGap(i) for all i;
        /* copy the entries of hyperplaneGap to the entries on the second axis of λ
           */
        λ(0, i) ← hyperplaneGap(i) for all i;
        λ(1, 1) ← 1;
        /* Now λ is partially filled and ready to be passed to fillGapFunction */
        fillGapFunction(λ, δ, (1, 1), gapFunList);
    end
end
if r > 2 then
    hListArray ← an array of size r, the i-th entry is an empty list of GapFunctions;
    /* The i-th list in hListArray holds a list of (r − 1)-dimensional GapFunctions
       that are potentially the coordinate hyperplanes λ_{H_i} of the GapFunction λ we
       are trying to produce                                                */
    add hyperplaneGap to the first list in hListArray;
    for i = 2 to r do
        iHyperplane ← the projection of hyperplaneGap onto its (i − 1)-th coordinate hyperplane;
        /* to produce λ_{H_i} we use recursion and assemble iHyperplaneGap and axisGap */
        Δ ← δ − (axisGap.degree) − 1;
        /* Δ is an upper bound for the degree of λ_{H_i}                   */
        for j = 0 to Δ do
            assembled ← assembleGapFunction(axisGap, iHyperplane, j);
            add the GapFunctions in assembled to the i-th list in hListArray;
        end
        if the i-th list in hListArray is empty for some i then
            /* axisGap and hyperplaneGap cannot be assembled together      */
            return an empty list;
        end
    end
    for every (λ_{H_1}, ..., λ_{H_r}) with λ_{H_i} in the i-th list of hListArray do
        if λ_{H_1}, ..., λ_{H_r} are compatible as projections of an r-dimensional GapFunction then
            λ ← a new GapFunction with dimension r and size (2(δ − 1) + 1, ..., 2(δ − 1) + 1);
            for i = 1 to r do
                fill the entries of λ on the i-th coordinate hyperplane with entries of λ_{H_i};
            end
            λ(1, 1, ..., 1) ← r − 1;
            /* λ is standard                                               */
            fillGapFunction(λ, δ, (1, 1, ..., 1),gapFunList);
        end
    end
end
```

3.2.4 Procedure `fillGapFunction`

This procedure (pseudocode on page 37) takes as input a partially filled `GapFunction` λ and a starting index $\alpha_{\text{start}} \in \mathbb{Z}^r_{>0}$ and tries to fill the rest of the cells. It assumes all the cells whose index has a zero component are filled. It also assumes the value at index α_{start} is filled. So the first cell to fill is increment(α_{start},λ.size). The procedure `fillGapFunction` uses a subprocedure called `shouldStop`. Every time we fill a cell of λ in `fillGapFunction`, the procedure `shouldStop` is called to check whether we are done filling the cells of λ (in which case `shouldStop` returns true) or not (when `shouldStop` returns false). We give the details of the procedure `shouldStop` in section 3.2.5.

The first task in `fillGapFunction` is to call `shouldStop` to make sure there is more work to do on λ (Line 1). If `shouldStop` returns true, we are done, otherwise we increment α_{start} in Line 6. The for loop in line 7 starts from $\alpha = \alpha_{\text{start}}$, after each iteration increments α using $\alpha \leftarrow$`increment`$(\alpha, \lambda.\text{size})$ until α reaches λ.size (inclusive). In each iteration the algorithm fills the cell of λ at position α. The filling is done using two properties of gap functions: First in Lines 8-17 we test whether the upward propagation property (Lemma 2.5.8), uniquely determines the value of λ at position α. If so, we fill in the cell α and call `shouldStop`. If `shouldStop` declares false, we go to the next iteration for α. If upward propagation does not determine the cell at α, we use Lemma 2.5.5 for determining $\lambda(\alpha)$ in Lines 18-35. There are two cases: 1) The lemma forces the value of α to be the same as the previous cell, in which case we are done with cell α. 2) The lemma cannot decide, in which case there are two possibilities for $\lambda(\alpha)$. Either $\lambda(\alpha) = \lambda(\alpha - e_1)$ or $\lambda(\alpha) = \lambda(\alpha - e_1) + 1$. Both possibilities are then pursued.

Procedure 4: fillGapFunction

input : λ (a GapFunction), δ (an integer), α_{start} (a list of integers), gapFunList (a list of GapFunctions)

output: a set of GapFunctions obtained by filling λ with δ in one of the cells (candidates for being gap functions obtained through the partially filled input λ); these GapFunctions are added to gapFunList

```
1  if shouldStop(λ,α,δ,gapFunList)=true then
2  |  return;
3  end
4  α_final ← λ.size;
5  d ← λ.dim;
6  α_start ← increment (α_start, α_final);
   /* In the next for loop, after each iteration the counter α is incremented by α ←
      increment(α, α_final). Comparison with the final value α_final is also done using ≼.
      */
7  for α from α_start to α_final do
   |  /* Check and see if λ(α) can be determined by upward propagation                  */
8  |  for every i, j such that i ≠ j do
9  |  |  if λ(..., α_i − 1, ..., α_j − 1, ...) < λ(..., α_i, α_j − 1, ...) then
10 |  |  |  λ(..., α_i, ..., α_j, ...) ← λ(..., α_i − 1, ..., α_j) + 1;
11 |  |  |  if shouldStop(λ,α,δ,gapFunList)=true then
12 |  |  |  |  return;
13 |  |  |  else
14 |  |  |  |  go to the next iteration for α;
15 |  |  |  end
16 |  |  end
17 |  end
   |  /* check Lemma 2.5.5                                                            */
   |  /* numOfOnes(α)= #{i|α_i = 1}                                                   */
18 |  if |α| > 2λ(α − e_1) + 2−numOfOnes(α) then
   |  |  /* λ(α) must be equal to λ(α − e_1)                                         */
19 |  |  λ(α) ← λ(α − e_1);
20 |  |  if shouldStop(λ,α,δ,gapFunList)=true then
21 |  |  |  return;
22 |  |  else
23 |  |  |  go to next iteration for α;
24 |  |  end
25 |  else
   |  |  /* λ(α) can be either λ(α − e_1) or λ(α − e_1) + 1                          */
26 |  |  λ' ← λ;
27 |  |  λ'(α) ← λ(α − e_1);
28 |  |  fillGapFunction(λ', δ, α,gapFunList);
29 |  |  λ(α) ← λ(α − e_1) + 1;
30 |  |  if shouldStop(λ,α,δ,gapFunList)=true then
31 |  |  |  return;
32 |  |  else
33 |  |  |  go to next iteration for α;
34 |  |  end
35 |  end
36 end
```

3.2.5 Procedure `shouldStop`

The aim of this procedure (pseudocode on page 39) is to decide whether we are done with filling a `GapFunction` λ. It takes as input λ, the index α of the cell that was just filled in, the degree δ that we are trying to reach and a list Λ of already computed `GapFunctions` with degree δ. The output is false if we decide that more cells of λ needs to be filled or true when we decide the process of filling is finished. In case the process is finished, the resulting λ is either a potential gap function (which is then printed and added to the list Λ) or it violates one of the properties and is not a valid gap function (which is then discarded). First we test whether the value of λ at the cell α is at most one more than $\lambda(\alpha - e_i)$ (see Lemma 2.5.1). If this property is violated λ is not valid and we return true and discard λ (by not adding it to Λ). Next, we test whether $\lambda(\alpha)$ is equal to δ. If this is not the case, we have not reached the degree that we aimed for and more cells of λ needs to be filled and we return false. If $\lambda(\alpha) = \delta$, then the process of filling λ is done. Before outputting λ (and adding it to the list Λ), two checks are done:

- λ needs to satisfy the semigroup property (Lemma 3.2.1), for all $1 \leq i \leq \lambda.\text{dim}$.
- λ needs to be a new table, i.e. no permutation of the components of λ is equal to a previously computed table in the list Λ.

If λ passes these two checks, we output λ, add it to the list Λ and return true.

Lemma 3.2.1 (Semigroup property)**.** *Suppose $\lambda : \mathbb{Z}^r_{\geq 0} \to \mathbb{Z}_{\geq 0}$ is a standard gap function. Then for each $1 \leq i \leq r$, the set*

$$\Sigma_i = \{\alpha \in \mathbb{Z}^r_{\geq 0} : \lambda(\alpha) = \lambda(\alpha + e_i)\},$$

is a semigroup.

Proof. If $\alpha, \alpha' \in \Sigma_i$, then by Lemma 2.5.1, $\alpha[i]$ and $\alpha'[i]$ belong to Σ. Thus, $(\alpha + \alpha')[i]$ belongs to Σ which gives $\alpha + \alpha' \in \Sigma_i$. □

Since a gap function is represented by a finite array, checking whether an array satisfies the semigroup property is done in a finite number of steps.

Procedure 5: shouldStop

input : λ (a GapFunction), α (a list of non-negative integers), δ (an integer), Λ (a list of GapFunctions)

output: false if the procedure of filling λ is not finished; true if the procedure of λ is done and if the obtained λ is not valid, it is discarded otherwise it is printed and also added to Λ

$d \leftarrow \lambda$.dim;
/* Check if from $\alpha - e_i$ to α, the value of λ increases by at most 1 */
for i *from* 1 *to* d **do**
  if $\lambda(\alpha - e_i) < \lambda(\alpha) - 1$ **then**
    return true;
  end
end
/* Check if $\lambda(\alpha) \neq \delta$ */
if $\lambda(\alpha) < \delta$ **then**
  return false;
end
/* Otherwise, we have $\lambda(\alpha) = \delta$ */
/* We need to check two properties: 1) Semigroup property, 2)Symmetry check. */
$\tilde{\lambda} \leftarrow$ a new GapFunction of size α with entries $\preccurlyeq \alpha$ copied from λ;
/* Check semiGroupProperty */
/* semigroupProperty($\tilde{\lambda}$,i) is a procedure that takes a GapFunction $\tilde{\lambda}$ and a direction i and validates Lemma 3.2.1 for $\tilde{\lambda}$ and i. The output is true if $\tilde{\lambda}, i$ satisfy the lemma, false otherwise. */
for $i = 1$ *to* d **do**
  if *semiGroupProperty*($\tilde{\lambda}, i$)=*false* **then**
    return true;
  end
end
/* Symmetry check */
permutationList$\leftarrow$ a list of all permutations of $(1, 2, \ldots, d)$;
for λ' *in* Λ **do**
  for π *in permutationList* **do**
    /* $\pi \cdot \tilde{\lambda}$ is a new GapFunction and is the result of permuting the components of the array $\tilde{\lambda}$ */
    if $\lambda' = \pi \cdot \tilde{\lambda}$ **then**
      /* $\tilde{\lambda}$ already exists in Λ */
      return true;
    end
  end
end
print $\tilde{\lambda}$;
add $\tilde{\lambda}$ to Λ;
return true;

3.3 Correctness of the algorithm

In this final section, we prove that the output of the procedure `main` is exactly what we want.

Proposition 3.3.1. *The output of the procedure* ***`main`*** *for input* $(\delta, r,$ ***`gapFunList`***$)$ *is a list of* r*-dimensional* ***`GapFunctions`*** *with degree* δ *such that the set of all* r*-dimensional gap functions with degree* δ *is subset of the output. Here we assume that* $\delta \geq 0$*,* $r \geq 1$ *and* ***`gapFunList`*** *contains all gap functions with degree less than* δ *and dimension less than* r*.*

Proof. We need to show that the algorithm terminates and that the output is correct. Termination is established by the finite nature of the tables we create and test. More specifically, given δ, r the algorithm tries to assemble a 1- and a $(r-1)$-dimensional gap functions with degree less than δ from the list `gapFunList`. The number of possibilities here is finite because `gapFunList` is a finite list. During each assembly, the algorithm tries to fill an r-dimensional `GapFunction` with size

$$(2(\delta-1)+1, \ldots, 2(\delta-1)+1)\,, \tag{3.1}$$

one entry at a time, which is done in a finite number of steps. The reason this size is chosen is because if a gap function does not attain the value δ by the entry (3.1), then its degree will be less than δ (see Lemma 2.5.4).

Now we explain the correctness of the algorithm. Because we are claiming that the output is a superset of the set of all gap functions with degree δ and dimension r, it is enough to show out of all possible `GapFunctions`, the ones that are discarded violate at least one property of gap functions. Note that the algorithm is recursive (in Procedure 3), hence we assume the output of the algorithm is correct for dimension less than r and degree less than δ.

Firstly in `main` and `assembleGapFunction`, we are producing `GapFunctions` whose projections onto their first axis and first coordinate hyperplane are gap functions in `gapFunList`. This discards `GapFunctions` that violate Lemma 2.6.2. Moreover, `GapFunctions` violating Lemma 2.6.3 are also discarded. These are all `GapFunctions` discarded in `main` and `assembleGapFunction`.

In `fillGapFunction` the only properties that are checked are upward propagation (Lemma 2.5.8) and Lemma 2.5.5. Therefore, this procedure discards those `GapFunctions` violating one of the two lemmas.

It remain to see what `GapFunctions` are discarded in `shouldStop`: If a `GapFunction` violates Lemma 2.5.1 it is discarded. After that the last filled entry is compared with δ. If it is less than δ, `shouldStop` signals `fillGapFunction` to continue filling entries, however if the last filled entry is equal to δ, then filling is considered done and the procedure checks the semigroup property (see Lemma 3.2.1) against the `GapFunction`. If the semigroup

property does not hold, the `GapFunction` is discarded otherwise it is compared with all `GapFunction`s with dimension r and degree δ that have been previously computed. If the current `GapFunction` is not a permutation of some previously computed table, then it is printed to the output and saved. □

At this point we have an algorithm that produces candidate gap functions. In the next chapter, we will focus on the output for $\delta = 4$ and show that every `GapFunction` in the output is indeed a gap function corresponding to a subalgebra $R \subset S$.

Chapter 4

Gap functions with degree 4

In this chapter we consider the output of the algorithm of Chapter 3 for $\delta = 4$, i.e. the gap functions with degree 4. We have implemented this algorithm in Maple. Appendix A contains the code of our implementation and its output for $\delta \leq 3$ is exactly the gap functions in Tables 2.1, 2.2, and 2.3. In Section 4.1, we list the output of the algorithm for $\delta = 4$ in tables. As we mentioned in Chapter 3, the set of gap functions with degree 4 is a subset of this output. In Section 4.2, we will show that each table of the output is a gap function by producing the corresponding $\mathbb{K}$-algebras $R \subset S$. In Section 4.3, we summarize our results and discuss future work.

4.1 Tables of gap functions with degree 4

Tables 4.1, 4.2, 4.3 and 4.4 list the r-dimensional `GapFunction`s that are the output of the algorithm in Chapter 3 for $\delta = 4$ and $1 \leq r \leq 4$. Each table is labeled with three numbers $\delta.r.n$: the first number δ denotes the degree of `GapFunction`, the second number r is the dimension of the table and the last part is the number of the table among tables with a given δ, r.

4.1.1:	**0**	**1**	1	**2**	2	**3**	3	**4** ...	**4.1.2:**	**0**	**1**	**2**	2	**3**	3	3	**4** ...
4.1.3:	**0**	**1**	**2**	2	**3**	**4** ...			**4.1.4:**	**0**	**1**	**2**	**3**	3	3	3	**4** ...
4.1.5:	**0**	**1**	**2**	**3**	3	3	**4** ...		**4.1.6:**	**0**	**1**	**2**	**3**	3	**4** ...		
4.1.7:	**0**	**1**	**2**	**3**	**4** ...												

Table 4.1: Gap functions with $r = 1$ and $\delta = 4$

In Table 4.3, a 3-dimensional `GapFunction` λ is presented as a sequence of 2-dimensional `GapFunction`s. A table in level $[i]$ contains the values $\lambda(\alpha_1, \alpha_2, i)$. Similarly, a 4-dimensional gap function λ' in Table 4.4 is presented as a sequence of 2-dimensional `GapFunction`s. The table at level $[i, j]$ contains the values $\lambda'(\alpha_1, \alpha_2, i, j)$.

4.2 Generators of subalgebras

In this section, we prove the main result of this thesis.

Theorem 4.2.1. *The standard gap functions with degree 4, up to permutation of the coordinates, are those listed in Tables 4.1, 4.2, 4.3, and 4.4. Furthermore, Tables 4.5, 4.6, 4.7, 4.8, and 4.9 list the corresponding algebras $R \subset S$.*

Proof. In Proposition 3.3.1, we saw that the set of all gap functions with degree 4 is a subset of the potential gap functions listed in Tables 4.1, 4.2, 4.3, and 4.4. It is enough now to produce subalgebras $R \subset S$ for each table. Tables 4.5, 4.6, 4.7, 4.8 and 4.9 list the generators for each table. We explain how we found these generators.

It is easy to find the generators for those potential gap functions λ with projections λ_{H_i} and λ_{L_i} that satisfy

$$4 = \delta(\lambda) = \delta(\lambda_{H_i}) + \delta(\lambda_{L_i}) + 1, \tag{4.1}$$

for some i. When this equality holds the potential gap function is uniquely determined from its projections via upward propagation (Lemma 2.5.8) and the generators for R can be obtained by taking the union of generators for λ_{H_i} and λ_{L_i} (Proposition 2.6.6). The tables that this equality holds for them are

- Tables **4.2.i** for $i \in \{8, 9, 10, 11, 15, 16\}$,
- Tables **4.3.j** for $j \in \{2, 7, 8, 9, 10, 11\}$,
- Tables **4.4.k** for $k \in \{2, 3, 4\}$.

For example consider the Table **4.2.15**. The projections of this potential gap function onto the axes are

$$\begin{array}{llll} \big|\, 0 & 1 & 1 & 2 \quad \ldots, \\ \big|\, 0 & 1 & 1 & \ldots, \end{array} \qquad \begin{array}{ll} \text{corresponding to} & R_1 = \mathbb{K}[[t_1^2, t_1^3]] \subset S = \mathbb{K}[[t_1]], \\ \text{corresponding to} & R_2 = \mathbb{K}[[t_2^2, t_2^5]] \subset S = \mathbb{K}[[t_2]]. \end{array}$$

The generators for R_1 and R_2 can be found in [BIV20, Tables 6, 7]. These projections have degrees 2 and 1 respectively and they satisfy (4.1). The subalgebra $R \subset S = \mathbb{K}[[t_1]] \times \mathbb{K}[[t_2]]$ that has Table **4.2.15** as gap function is then

$$R = \mathbb{K}[[t_1^2, t_1^3, t_2^2, t_2^5]].$$

For the tables that are not product of lower degree gap functions, we need to do more work. The general argument is as in [BIV20, Proposition 5.4]. We describe the procedure and then give several examples. The procedure is the following 4 steps:

1. Given a table λ, we write down certain elements in Σ by using Lemma 2.5.1 and Lemma 2.5.6.

2. For each element σ of Σ in the previous step, we write down an element f_σ in R. We use automorphisms of S and certain elements in R to make f_σ as simple as possible.

3. We set R to be the subalgebra inside $S = \prod_{i=1}^{r} \mathbb{K}[[t_i]]$ (here r is the dimension of λ) generated by all f_σ we found in the previous step. An explicit computation then shows that the completion R' of R is such that $\lambda_{R'} = \lambda$.

4. We can use a computer algebra system (such as Macaulay2) to compute the relations among the generators of R. This then gives R' as a quotient of S.

□

Example 4.2.2. Consider Table **4.1.1**. We want to find the generators of $R \subset \mathbb{K}[[t]]$ for this table and show that this is in fact a gap function. We can see by Lemma 2.5.1 that

$$\begin{aligned}\lambda(2) = \lambda(3) &\to 2 \in \Sigma,\\ \lambda(4) = \lambda(5) &\to 4 \in \Sigma,\\ \lambda(6) = \lambda(7) &\to 6 \in \Sigma,\end{aligned}$$

and

$$\lambda(8) = \lambda(9) = \lambda(10) = \ldots \quad \to k \in \Sigma \quad \text{for} \quad k \geq 8.$$

Hence, there exists a function $x \in R$ such that $\nu(x) = 2$. After applying an automorphism of $\mathbb{K}[[t]]$, we may assume that $x = t^2$. There is also $y \in R$ such that $\nu(y) = 9$. Because $i \in \Sigma$ for $i \geq 8$, by subtracting suitable elements in R from y we can assume $y = t^9$. The generators for this gap function is $R = \mathbb{K}[[t^2, t^9]]$. To compute the relations among the generators (e.g in Macaulay2), we can eliminate t in the ideal $\langle x - t^2, y - t^9 \rangle \subset \mathbb{K}[x, y, t]$. This results in the ideal $\langle x^9 - t^2 \rangle$. The completion of R then has a presentation $R \cong \mathbb{K}[[x, y]]/(x^9 - y^2)$.

Example 4.2.3. Consider Table **4.2.7**. By Lemma 2.5.1, we have

$$\begin{aligned}\lambda(1,1) = \lambda(1,2) \Rightarrow (1,1)[2] \in \Sigma \Rightarrow (m,1) \in \Sigma \quad &\text{for some} \quad 4 > m \geq 1,\\ \lambda(2,1) = \lambda(2,2) \Rightarrow (2,1)[2] \in \Sigma \Rightarrow (n,1) \in \Sigma \quad &\text{for some} \quad 4 > n \geq 2,\\ \lambda(3,1) = \lambda(3,2) \Rightarrow (3,1)[2] \in \Sigma \Rightarrow (p,1) \in \Sigma \quad &\text{for some} \quad 4 > p \geq 3.\end{aligned}$$

By the above inequalities, we conclude the following $(3,1) \in \Sigma$ and after an automorphism of $S = \mathbb{K}[[t_1]] \times \mathbb{K}[[t_2]]$ we may assume $t_1^3 + t_2 \in R$. In addition, we have $(\infty, k) \in \Sigma$ for $k \geq 2$, and $(k, \infty) \in \Sigma$ for $k \geq 4$. Now we can write the generators as follows: $R = \mathbb{K}[[t_1^4, t_1^5, t_1^6, t_1^7, t_2^2, t_2^3, t_1^3 + t_2]]$. To compute the relations among the generators, denote the seven generators of R that we just found by $g_1, \ldots, g_7$. Then eliminate t_1, t_2 from the ideal $\langle x_1 - g_1, \ldots, x_7 - g_7, t_1 t_2 \rangle \subset \mathbb{K}[x_1, \ldots, x_7, t_1, t_2]$. Note that here we are including $t_1 t_2$ because of the relation $t_1 t_2 = 0$.

Example 4.2.4. Let's consider Table **4.3.4**. Here $R \subset S = \prod_{i=1}^{3} \mathbb{K}[[t_i]]$. Since the size of this table is $(2, 4, 2)$, we can conclude that

$$\begin{aligned}(2,\infty,\infty),(3,\infty,\infty) &\in \Sigma,\\ (\infty,4,\infty),(\infty,5,\infty),(\infty,6,\infty) &\in \Sigma,\\ (\infty,\infty,2),(\infty,\infty,3) &\in \Sigma.\end{aligned}$$

By using Lemma 2.5.1, we can see the following:

$$(1,2,1) \in \Sigma, \quad (1,3,1) \in \Sigma, \quad (1,4,1) \in \Sigma,$$

which will show that $(1,\infty,1) \in \Sigma$, and after an automorphism of S we have $t_1 + t_3 \in R$. Also by subtracting powers of $t_1 + t_3$ from the element in R with valuation $(2,\infty,\infty)$, we conclude $t_1^2 \in R$.

Also, we have $(2,2,1),(2,3,1) \in \Sigma$. Moreover, by Lemma 2.5.6 with $\alpha = (1,2,0), i = 2$, we have

$$u' t_2^2 \in R \Rightarrow (\infty,2,\infty) \in \Sigma.$$

Now, by computation in the semigroup Σ, we have:

$$\begin{aligned}(1,3,1) + (\infty,2,\infty) &= (\infty,5,\infty) \in \Sigma,\\ (2,3,1) + (\infty,\infty,2) &= (\infty,\infty,3) \in \Sigma.\end{aligned}$$

So $(1,3,1),(\infty,2,\infty)$ already generate $(\infty,4,\infty),(\infty,5,\infty),(\infty,6,\infty),(\infty,7,\infty)$ in Σ.

Furthermore,

$$\begin{aligned}(1,2,1) \in \Sigma &\Rightarrow (t_1, \beta_2 t_2^2 + \beta_3 t_2^3, \gamma_1 t_3) \in R\\ (1,3,1) \in \Sigma &\Rightarrow (t_1, \beta_2' t_2^3, \gamma_1' t_3) \in R\\ (2,2,1) \in \Sigma &\Rightarrow (t_1^2, \beta_2'' t_2^2 + \beta_3'' t_2^3, \gamma_1'' t_3) \in R.\end{aligned}$$

Since we have $(2,\infty,\infty),(\infty,2,\infty) \in \Sigma$, we can eliminate the components $t_1^2, \beta_2 t_2^2$, and $\beta_2'' t_2^2$. Therefore, we have

$$\begin{aligned}(t_1, 0, \gamma_1 t_3) &\in R,\\ (0, 0, \gamma_1'' t_3) &\in R.\end{aligned}$$

Also, since $(2,3,1) \in \Sigma$, we can conclude that $(0, \beta_3''' t_2^3, \gamma_1''' t_3) \in R$, which finally brings us to $(0, t_2^3, \gamma_1''' t_3) \in R$. Hence, after an automorphism of S, the generators are $R = \mathbb{K}[[t_1^2, t_1 + t_3, t_2^2, t_2^3 + t_3]]$. Relations among the generators are computed similar to the previous examples.

Example 4.2.5. Consider gap function 4.4.1. Since the size of the function is $(2,2,2,2)$, we have

$$\begin{aligned}(2,\infty,\infty,\infty),(3,\infty,\infty,\infty)\in\Sigma,\\(\infty,2,\infty,\infty),(\infty,3,\infty,\infty)\in\Sigma,\\(\infty,\infty,2,\infty),(\infty,\infty,3,\infty)\in\Sigma,\\(\infty,\infty,\infty,2),(\infty,\infty,\infty,3)\in\Sigma.\end{aligned}$$

By looking at the table and considering Lemma 2.5.1,

$$\begin{aligned}(1,1,1,1),(1,1,2,1)\in\Sigma,\\(1,1,1,2),(1,1,2,2)\in\Sigma,\\(2,1,1,1),(2,1,2,1)\in\Sigma,\\(2,1,1,2),(1,2,1,1)\in\Sigma,\\(1,2,2,1),(1,2,1,2)\in\Sigma,\\(2,2,1,1)\in\Sigma.\end{aligned}$$

Thus, we can conclude that after a suitable automorphism of $S=\prod_{i=1}^{4}\mathbb{K}[[t_i]]$

$$\begin{aligned}(t_1,t_2,t_3,t_4)\in R,\\(t_1,\beta_1 t_2,0,\eta_1 t_4)\in R,\\(t_1,\beta_1' t_2,\gamma_1 t_3,0)\in R,\\(t_1,\beta_1'' t_2,0,0)\in R,\\(0,t_2,\gamma_1' t_3,\eta_1' t_4)\in R,\\(0,t_2,0,\eta_1'' t_4)\in R,\\(0,t_2,\gamma_1'' t_3,0)\in R,\\(t_1,0,\gamma_1''' t_3,\eta_1''' t_4)\in R,\\(t_1,0,0,\eta_1'''' t_4)\in R,\\(t_1,0,\gamma_1''''' t_3,0)\in R,\\(0,0,t_3,\eta_1''''' t_4)\in R.\end{aligned}$$

We want to show that $(t_1,0,0,\eta_1'''' t_4)$, $(0,t_2,0,\eta_1'' t_4)$, and $(0,0,t_3,\eta''''' t_4)$ are enough. Because

$$V=\{(\alpha t_1,\beta t_2,\gamma t_3,\eta t_4)|\alpha,\beta,\gamma,\eta\in\mathbb{K}\}$$

is a vector space of dimension 4, there can be at most 4 independent vectors. Those three vectors are linearly independent. We now have two cases: either the span of all the elements cover the whole 4 dimensional space (V) or their span is 3 dimensional. We claim that the

span of these elements cannot be 4 dimensional.

$$V = \text{span}\{(t_1, 0, 0, 0), (0, t_2, 0, 0), (0, 0, t_3, 0), (0, 0, 0, t_4)\}.$$

If the span is 4-dimensional, then $t_1, t_2, t_3, t_4 \in R$.

$$\lambda(\alpha_1, \alpha_2, \alpha_3, \alpha_4) = \dim \frac{K[[t_1]] \times \cdots \times K[[t_4]]}{R + \langle t_1^{\alpha_1}, t_2^{\alpha_2}, t_3^{\alpha_3}, t_4^{\alpha_4} \rangle}.$$

So we have a gap function that $\lambda(1, 1, 1, 1) = \delta = 3$ and it does not change after that cell. For $\alpha_i \geq 1, \langle t_1^{\alpha_1}, t_2^{\alpha_2}, t_3^{\alpha_3}, t_4^{\alpha_4} \rangle \subset R = K[[t_1, t_2, t_3, t_4]]$. Hence, it does not change from $(\alpha_1, \alpha_2, \alpha_3, \alpha_4) = (1, 1, 1, 1)$ onwards:

$$\lambda(\alpha_1, \ldots, \alpha_4) = \dim \frac{S}{R} = 3 \Rightarrow \delta = 3,$$

which is a contradiction. Therefore, it has to be 3-dimensional. So these three vectors are enough:

$$(t_1, 0, 0, \eta_1''''t_4), (0, t_2, 0, \eta_1''t_4), (0, 0, t_3, \eta_1'''''t_4).$$

Therefore, $R = K[[t_1 + \eta_1 t_4, t_2 + \eta_2 t_4, t_3 + \eta_3 t_4, t_1^2, t_1^3, t_2^2, t_2^3, t_3^2, t_3^3]]$. Note that this R is not necessarily unique up to automorphisms of S because we have two parameters η_1, η_2 in the generators. Among all gap functions with degree 4 there are two, namely **4.4.1** and **4.3.1**, whose algebras are not necessarily unique up to automorphisms of their normalizations.

4.3 Analysis of results and future work

Our work in this thesis shows that there are 39 standard gap functions with degree 4. A summary of the number of gap functions with degree at most 5 is (the last row comes from the output of our algorithm for $\delta = 5$, see Appendix B):

Degree	Number of gap functions
1	2
2	4
3	14
4	39
5	≤ 119

A distinct feature of $\delta = 4$ is that there might exist non-isomorphic subalgebras $R \subset S$ whose gap functions are the same. This does not happen when $\delta \leq 3$. If λ is a gap function with degree δ for which there are non-isomorphic subalgebras, then by taking the product

of λ with the one dimensional zero gap function, we produce a gap function with degree $\delta + 1$ that has non-isomorphic subalgebras. In this way non-uniqueness of the subalgebras will propagate to higher degrees.

The following is a list of questions for future work:

1. We have seen that our algorithm is correct (produces only gap functions) when the degree is at most 4. Is this algorithm correct for higher degrees?

2. How many gap functions with degree 5 are there? Appendix B contains the output of our algorithm for $\delta = 5$. Are all of them gap functions?

3. Does the sequence of the number of gap functions with degree δ (i.e. $2, 4, 14, 39, \ldots$) follow a specific pattern? Can we give an upper bound for the number of gap functions with a given degree?

4. Are there non-isomorphic subalgebras $R_1, R_2 \subset S$ whose gap functions are equal to the gap function **4.3.1**? What about the gap function **4.4.1**? We conjecture that non-isomorphic R_1, R_2 with this property exist for both tables.

4.2.1:

⋮	⋮	⋮	⋮	
1	2	3	**4**	…
1	2	**3**	3	…
1	**2**	2	2	…
1	1	1	1	…

4.2.2:

⋮	⋮	⋮	⋮	⋮	
1	2	3	3	**4**	…
1	2	**3**	3	3	…
1	**2**	2	2	2	…

4.2.3

⋮	⋮	⋮	⋮	⋮	⋮	
1	2	2	3	3	**4**	…
1	**2**	2	**3**	3	3	…

4.2.4:

⋮	⋮	⋮	⋮	
1	2	3	**4**	…
1	**2**	2	**3**	…

4.2.5:

⋮	⋮	⋮	⋮	⋮	⋮	
1	2	3	3	3	**4**	…
1	**2**	**3**	3	3	3	…

4.2.6:

⋮	⋮	⋮	⋮	⋮	
1	2	3	3	**4**	…
1	**2**	**3**	3	3	…

4.2.7:

⋮	⋮	⋮	⋮	
1	2	3	**4**	…
1	**2**	**3**	3	…

4.2.8:

⋮	⋮	⋮	⋮	⋮	⋮	
1	**2**	2	**3**	3	**4**	…

4.2.9:

⋮	⋮	⋮	⋮	⋮	⋮	
1	**2**	**3**	3	3	**4**	…

4.2.10:

⋮	⋮	⋮	⋮	⋮	
1	**2**	**3**	3	**4**	…

4.2.11:

⋮	⋮	⋮	⋮	
1	**2**	**3**	**4**	…

4.2.12:

⋮	⋮	⋮	⋮	
2	3	3	**4**	…
2	3	3	3	…
2	**3**	3	3	…
1	**2**	2	2	…

4.2.13:

⋮	⋮	⋮	
2	3	**4**	…
2	3	3	…
2	**3**	3	…
1	**2**	2	…

4.2.14:

⋮	⋮	⋮	
2	3	**4**	…
2	**3**	3	…
1	**2**	2	…

4.2.15:

⋮	⋮	⋮	⋮	
2	3	3	**4**	…
1	**2**	2	**3**	…

4.2.16:

⋮	⋮	⋮	
2	3	**4**	…
1	**2**	**3**	…

Table 4.2: Gap functions with $r = 2$ and $\delta = 4$. Product gap functions are underlined.

4.3.1†**:**

Level [1]

⋮	⋮	⋮	
2	2	2	…
2	2	2	…

Level [2]

⋮	⋮	⋮	
2	3	3	…
2	**3**	3	…

Level [3]

⋮	⋮	⋮	
2	3	**4**	…
2	3	3	…

4.3.2:

Level [1]

⋮	⋮	⋮	
2	2	2	…

Level [2]

⋮	⋮	⋮	
2	**3**	3	…

Level [3]

⋮	⋮	⋮	
2	3	**4**	…

4.3.3:

Level [1]

⋮	⋮	
2	**3**	…
2	2	…

Level [2]

⋮	⋮	
3	**4**	…
2	3	…

4.3.4:

Level [1]

⋮	⋮	
3	3	…
3	3	…
3	3	…
2	2	…

Level [2]

⋮	⋮	
3	**4**	…
3	3	…
3	3	…
2	2	…

4.3.5:

Level [1]

⋮	⋮	
3	3	…
3	3	…
2	2	…

Level [2]

⋮	⋮	
3	**4**	…
3	3	…
2	2	…

4.3.6:

Level [1]

⋮	⋮	
3	3	…
2	2	…

Level [2]

⋮	⋮	
3	**4**	…
2	3	…

4.3.7:

Level [1]

⋮	⋮	
3	**4**	…
3	3	…
3	3	…
2	2	…

4.3.8:

Level [1]

⋮	⋮	
3	**4**	…
3	3	…
2	2	…

4.3.9:

Level [1]

⋮	
4	…
3	…
3	…
2	…

4.3.10:

Level [1]

⋮	
4	…
3	…
2	…

4.3.11:

Level [1]

⋮	
3	…
2	…

Level [2]

⋮	
4	…
3	…

Table 4.3: Gap functions with $r = 3$ and $\delta = 4$. Product gap functions are underlined. The symbol † denotes a gap function whose algebra is not necessarily unique.

4.4.1†**:**

Level $[1,1]$

$\vdots$	$\vdots$	
3	3	$\dots$
3	3	$\dots$

Level $[2,1]$

$\vdots$	$\vdots$	
3	3	$\dots$
3	3	$\dots$

Level $[1,2]$

$\vdots$	$\vdots$	
3	3	$\dots$
3	3	$\dots$

Level $[2,2]$

$\vdots$	$\vdots$	
3	**4**	$\dots$
3	3	$\dots$

4.4.2:

Level $[1,1]$

$\vdots$	$\vdots$	
3	3	$\dots$

Level $[2,1]$

$\vdots$	$\vdots$	
3	3	$\dots$

Level $[1,2]$

$\vdots$	$\vdots$	
3	3	$\dots$

Level $[2,2]$

$\vdots$	$\vdots$	
3	**4**	$\dots$

4.4.3:

Level $[1,1]$

$\vdots$	$\vdots$	
3	3	$\dots$

Level $[1,2]$

$\vdots$	$\vdots$	
3	**4**	$\dots$

4.4.4:

Level $[1,1]$

$\vdots$	
3	$\dots$

Level $[2,1]$

$\vdots$	
4	$\dots$

4.5.1: $\lambda(1,1,1,1,1) = 4$

Table 4.4: Gap functions with $r = 4, 5$ and $\delta = 4$. Product gap functions are underlined. The symbol † denotes a gap function whose algebra is not necessarily unique.

λ	Elements in Σ	Generators of R	Relations
4.1.1	2, 9	$x = t_1^2$, $y = t_1^9$	$x^9 - y^2$
4.1.2	3, 5, 6	$x = t_1^3$, $y = t_1^5$	$x^5 - y^3$
4.1.3	3, 7, 8	$x = t_1^3$, $y = t_1^7$, $z = t_1^8$	$x^2z - y^2$, $xy^3 - z^3$, $x^3y - z^2$, $y^5 - xz^4$, $x^5 - yz$
4.1.4	4, 5, 6	$x = t_1^4$, $y = t_1^5$, $z = t_1^6$	$x^3 - z^2$, $y^2 - xz$
4.1.5	4, 5, 7	$x = t_1^4$, $y = t_1^5$, $z = t_1^7$	$y^3 - x^2z$, $xy^2 - z^2$, $x^3 - yz$
4.1.6	4, 6, 7 9	$x = t_1^4$, $y = t_1^6$, $z = t_1^7$, $w = t_1^9$	$yz - xw$, $z^3 - y^2w$, $xz^2 - w^2$, $x^2z - yw$, $y^3 - w^2$, $xy^2 - zw$, $x^2y - z^2$, $x^3 - y^2$
4.1.7	5, 6, 7 8, 9	$x = t_1^5$, $y = t_1^6$, $z = t_1^7$, $w = t_1^8$, $q = t_1^9$	$w^2 - zq$, $zw - yq$, $yw - xq$, $z^2 - xq$, $yz - xw$, $y^2 - xz$, $x^2w - q^2$, $x^2z - wq$, $x^2y - zq$, $x^3 - yq$

Table 4.5: Generators for gap functions with $\delta = 4$, $r = 1$.

λ	Elements in Σ	Generators of R	(Number of) Relations
4.2.1	$(4,\infty)$, $(5,\infty)$, $(6,\infty)$, $(7,\infty)$, $(\infty,4)$, $(\infty,5)$, $(\infty,6)$, $(\infty,7)$, $(1,1)$, $(2,2)$, $(3,3)$	$x_1 = t_1^4$, $x_2 = t_1^5$, $x_3 = t_1^6$, $x_4 = t_1^7$, $y_1 = t_2^4$, $y_2 = t_2^5$, $y_3 = t_2^6$, $y_4 = t_2^7$, $z_1 = t_1 + t_2$, $z_2 = t_1^2 + t_2^2$, $z_3 = t_1^3 + t_2^3$	52
4.2.2	$(3,\infty)$, $(5,\infty)$, $(7,\infty)$, $(\infty,3)$, $(\infty,4)$, $(\infty,5)$, $(2,1)$	$x_1 = t_1^3$, $x_2 = t_1^5$, $x_3 = t_1^7$, $y_1 = t_2^3$, $y_2 = t_2^4$, $y_3 = t_2^5$, $z = t_1^2 + t_2$	19
4.2.3	$(2,\infty)$, $(7,\infty)$, $(\infty,2)$, $(\infty,3)$, $(5,1)$	$x_1 = t_1^2$, $x_2 = t_1^7$, $y_1 = t_2^2$, $y_2 = t_2^3$, $z = t_1^5 + t_2$	11
4.2.4	$(4,\infty)$, $(5,\infty)$, $(6,\infty)$, $(7,\infty)$, $(\infty,2)$, $(\infty,3)$, $(2,1)$	$x_1 = t_1^4$, $x_2 = t_1^5$, $x_3 = t_1^6$, $x_4 = t_1^7$, $y_1 = t_2^2$, $y_2 = t_2^3$, $z = t_1^2 + t_2$	18
4.2.5	$(3,\infty)$, $(4,\infty)$, $(\infty,2)$, $(5,1)$	$x_1 = t_1^3$, $x_2 = t_1^4$, $y_1 = t_2^2$, $y_2 = t_z^3$, $z = t_1^5 + t_2$	9
4.2.6	$(3,\infty)$, $(5,\infty)$, $(7,\infty)$, $(\infty,2)$, $(\infty,3)$, $(4,1)$	$x_1 = t_1^3$, $x_2 = t_1^5$, $x_3 = t_1^7$, $y_1 = t_2^2$, $y_2 = t_2^3$, $z = t_1^4 + t_2$	12
4.2.7	$(4,\infty)$, $(5,\infty)$, $(6,\infty)$, $(7,\infty)$, $(\infty,2)$, $(\infty,3)$, $(3,1)$	$x_1 = t_1^4$, $x_2 = t_1^5$, $x_3 = t_1^6$, $x_4 = t_1^7$, $y_1 = t_2^2$, $y_2 = t_2^3$, $z = t_1^3 + t_2$	18
<u>4.2.8</u>	$(\infty,2), (\infty,7)$	$x = t_1$, $y_1 = t_2^2$, $y_2 = t_2^7$	xy_2, xy_1, $y_1^7 - y_2^2$
<u>4.2.9</u>	$(\infty,3), (\infty,4)$	$x = t_1$, $y_1 = t_2^3$, $y_2 = t_2^4$	xy_2, xy_1, $y_1^4 - y_2^3$
<u>4.2.10</u>	$(\infty,3), (\infty,5)$, $(\infty,7)$	$x = t_1$, $y_1 = t_2^3$, $y_2 = t_2^5$, $y_3 = t_2^7$	6
<u>4.2.11</u>	$(\infty,4), (\infty,5)$, $(\infty,6), (\infty,7)$	$x = t_1$, $y_1 = t_2^4$, $y_2 = t_2^5$, $y_3 = t_2^6$, $y_4 = t_2^7$	10
4.2.12	$(2,\infty)$, $(5,\infty)$, $(\infty,2)$, $(\infty,5)$, $(3,3)$	$x_1 = t_1^2$, $x_2 = t_1^5$, $y_1 = t_2^2$, $y_2 = t_2^5$, $z = t_1^3 + t_2^3$	8

Table 4.6: Generators for gap functions with $\delta = 4$, $r = 2$. Product gap functions are underlined.

λ	Elements in Σ	Generators of R	(Number of) Relations
4.2.13	$(2,\infty)$, $(5,\infty)$ $(\infty,3)$, $(\infty,4)$, $(\infty,5)$, $(3,2)$	$x_1 = t_1^2$, $x_2 = t_1^5$, $y_1 = t_2^3$, $y_2 = t_2^4$, $y_3 = t_2^5$, $z = t_1^3 + t_2^2$	13
4.2.14	$(3,\infty)$, $(4,\infty)$, $(5,\infty)$, $(\infty,3)$, $(\infty,4)$, $(\infty,5)$, $(2,2)$	$x_1 = t_1^3$, $x_2 = t_1^4$, $x_3 = t_1^5$, $y_1 = t_2^3$, $y_2 = t_2^4$, $y_3 = t_2^5$ $z = t_1^2 + t_2^2$	18
<u>4.2.15</u>	$(2,\infty), (3,\infty)$, $(\infty,2), (\infty,5)$	$x_1 = t_1^2$, $x_2 = t_1^3$, $y_1 = t_2^2$, $y_2 = t_2^5$	6
<u>4.2.16</u>	$(2,\infty), (3,\infty)$, $(\infty,3), (\infty,4)$, $(\infty,5)$	$x_1 = t_1^2$, $x_2 = t_1^3$, $y_1 = t_2^3$, $y_2 = t_2^4$, $y_3 = t_2^5$	10

Table 4.7: Generators for gap functions with $\delta = 4$, $r = 2$ (continued). Product gap functions are underlined.

λ	Elements in Σ	Generators of R	(Number of) Relations
4.3.1†	$(1,\infty,1)$, $(\infty,1,2)$, $(2,1,\infty)$, $(\infty,\infty,3)$	$x_1 = t_1^2 + t_2$, $x_2 = t_2 + at_3^2$, $x_3 = t_1 + t_3 + bt_3^2$, $y_1 = t_3^3$	10
4.3.2	$(1,1,\infty)$, $(3,\infty,\infty)$, $(\infty,\infty,1)$	$x = t_1 + t_2$, $y = t_1^3$, $z = t_3$	yz, xz, $x^3y - y^2$
4.3.3	$(2,\infty,\infty)$, $(\infty,2,\infty)$, $(1,1,1)$	$x = t_1^2$, $y = t_2^2$, $z = t_1 + t_2 + t_3$	xy, $yz^2 - y^2$, $xz^2 - x^2$
4.3.4	$(1,\infty,1)$, $(\infty,3,1)$, $(2,\infty,\infty)$, $(\infty,2,\infty)$	$x = t_1 + t_3$, $y = t_2^3 + t_3$, $z = t_1^2$, $w = t_2^2$	5
4.3.5	$(1,\infty,1)$, $(\infty,2,1)$, $(2,\infty,\infty)$, $(\infty,3,\infty)$, $(\infty,4,\infty)$, $(\infty,\infty,2)$	$x = t_1 + t_3$, $y = t_2^2 + t_3$, $z = t_1^2$, $w_1 = t_2^3$, $w_2 = t_2^4$, $p = t_3^2$	13
4.3.6	$(1,\infty,1)$, $(2,\infty,\infty)$, $(\infty,2,\infty)$, $(\infty,3,\infty)$	$x = t_1 + t_3$, $y = t_1^2$, $z_1 = t_2^2$, $z_2 = t_2^3$	6
4.3.7	$(3,1,\infty)$, $(2,\infty,\infty)$, $(\infty,\infty,1)$	$x = t_1^3 + t_2$, $y = t_1^2$, $z = t_3$	yz, xz, $y^4 - x^2y$
4.3.8	$(2,1,\infty)$, $(3,\infty,\infty)$, $(\infty,2,\infty)$, $(\infty,\infty,1)$	$x = t_1^2 + t_2$, $y = t_1^3$, $z = t_2^2$, $w = t_3$	6
4.3.9	$(2,\infty,\infty)$, $(5,\infty,\infty)$ $(\infty,1,\infty)$, $(\infty,\infty,1)$	$x_1 = t_1^2$, $x_2 = t_1^5$, $y = t_2$, $z = t_3$	6
4.3.10	$(3,\infty,\infty)$, $(4,\infty,\infty)$, $(5,\infty,\infty)$, $(\infty,1,\infty)$, $(\infty,\infty,1)$	$x_1 = t_1^3$,$x_2 = t_1^4$, $x_3 = t_1^5$, $y = t_2$, $z = t_3$	10
4.3.11	$(2,\infty,\infty)$, $(3,\infty,\infty)$, $(\infty,1,\infty)$, $(\infty,\infty,2)$, $(\infty,\infty,3)$	$x_1 = t_1^2$, $x_2 = t_1^3$, $y = t_2$, $z_1 = t_3^2$,$z_2 = t_3^3$	10

Table 4.8: Generators for gap functions with $\delta = 4$, $r = 3$. Product gap functions are underlined. The symbol † denotes a gap function whose algebra is not necessarily unique.

λ	Elements in Σ	Generators of R	(Number of) Relations
4.4.1†	$(1,\infty,\infty,1)$, $(\infty,1,\infty,1)$, $(\infty,\infty,1,1)$, $(2,\infty,\infty,\infty)$, $(3,\infty,\infty,\infty)$, $(\infty,2,\infty,\infty)$, $(\infty,3,\infty,\infty)$, $(\infty,\infty,2,\infty)$, $(\infty,\infty,3,\infty)$	$x_1 = t_1 + \eta_1 t_4$, $x_2 = t_2 + \eta_2 t_4$, $x_3 = t_3 + \eta_3 t_4$, $y_1 = t_1^2$, $y_2 = t_1^3$, $z_1 = t_2^2$, $z_2 = t_2^3$, $w_1 = t_3^2$, $w_2 = t_3^3$	38
<u>4.4.2</u>	$(1,1,\infty,\infty)$, $(1,\infty,1,\infty)$, $(\infty,\infty,\infty,1)$	$x = t_1 + t_2$, $y = t_1 + t_3$, $z = t_4$	yz, xz, $x^2y - xy^2$
<u>4.4.3</u>	$(\infty,1,1,\infty)$, $(1,\infty,\infty,\infty)$, $(\infty,2,\infty,\infty)$, $(\infty,\infty,\infty,1)$	$x = t_2 + t_3$, $y = t_1$, $z = t_2^2$, $w = t_4$	6
<u>4.4.4</u>	$(1,\infty,\infty,\infty)$, $(\infty,1,\infty,\infty)$, $(\infty,\infty,2,\infty)$, $(\infty,\infty,3,\infty)$, $(\infty,\infty,\infty,1)$	$x = t_1$, $y = t_2$, $z_1 = t_3^2$, $z_2 = t_3^3$ $w = t_4$	10

Table 4.9: Generators for gap functions with $\delta = 4$, $r = 4$. Product gap functions are underlined. The symbol † denotes a gap function whose algebra is not necessarily unique.

Bibliography

[AM69] M. F. Atiyah and I. G. Macdonald. *Introduction to commutative algebra*. Addison-Wesley Publishing Co., Reading, Mass.-London-Don Mills, Ont., 1969, pp. ix+128.

[BIV20] Jarosław Buczyński, Nathan Ilten, and Emanuele Ventura. "Singular curves of low degree and multifiltrations from osculating spaces". In: *Int. Math. Res. Not. IMRN* 21 (2020), pp. 8139–8182. ISSN: 1073-7928. DOI: 10.1093/imrn/rnaa009.

[Cot05] Ethan Cotterill. "Rational curves of degree 10 on a general quintic threefold". In: *Communications in Algebra* 33.6 (2005), pp. 1833–1872.

[DM69] P. Deligne and D. Mumford. "The irreducibility of the space of curves of given genus". In: *Inst. Hautes Études Sci. Publ. Math.* 36 (1969), pp. 75–109. ISSN: 0073-8301. URL: http://www.numdam.org/item?id=PMIHES_1969__36__75_0.

[DH88] Steven Diaz and Joe Harris. "Geometry of the Severi variety". In: *Trans. Amer. Math. Soc.* 309.1 (1988), pp. 1–34. ISSN: 0002-9947. DOI: 10.2307/2001156. URL: https://doi-org.proxy.lib.sfu.ca/10.2307/2001156.

[Eis95] David Eisenbud. *Commutative algebra*. Vol. 150. Graduate Texts in Mathematics. With a view toward algebraic geometry. Springer-Verlag, New York, 1995, pp. xvi+785. ISBN: 0-387-94268-8; 0-387-94269-6. DOI: 10.1007/978-1-4612-5350-1. URL: https://doi-org.proxy.lib.sfu.ca/10.1007/978-1-4612-5350-1.

[Har77] Robin Hartshorne. *Algebraic geometry*. Graduate Texts in Mathematics, No. 52. Springer-Verlag, New York-Heidelberg, 1977, pp. xvi+496. ISBN: 0-387-90244-9.

[IM21] Nathan Ilten and Ahmad Mokhtar. "Khovanskii-finite rational curves of arithmetic genus 2". In: *arXiv preprint arXiv:2101.09410* (2021).

[JK96] Trygve Johnsen and Steven L. Kleiman. "Rational curves of degree at most 9 on a general quintic threefold". In: *Comm. Algebra* 24.8 (1996), pp. 2721–2753. ISSN: 0092-7872. DOI: 10.1080/02560049608542652. URL: https://doi-org.proxy.lib.sfu.ca/10.1080/02560049608542652.

[Kat86] Sheldon Katz. "On the finiteness of rational curves on quintic threefolds". In: *Compositio Math.* 60.2 (1986), pp. 151–162. ISSN: 0010-437X. URL: http://www.numdam.org/item?id=CM_1986__60_2_151_0.

[Kun05] Ernst Kunz. *Introduction to plane algebraic curves*. Translated from the 1991 German edition by Richard G. Belshoff. Birkhäuser Boston, Inc., Boston, MA, 2005, pp. xiv+293. ISBN: 978-0-8176-4381-2; 0-8176-4381-8.

[Map19] Maplesoft, a division of Waterloo Maple Inc.. *Maple*. Version 2019. Waterloo, Ontario, 2019. URL: https://hadoop.apache.org.

[Nam84] Makoto Namba. *Geometry of projective algebraic curves*. Vol. 88. Monographs and Textbooks in Pure and Applied Mathematics. Marcel Dekker, Inc., New York, 1984, pp. x+409. ISBN: 0-8247-7222-9.

Appendix A

Implementation of algorithm in Maple

This appendix contains our implementation of the algorithm for producing candidate gap functions with a given degree δ in Maple. First the code in Section A.1 must be executed. Then, to compute r-dimensional gap functions with degree δ, we use the following block of code. Here we are setting $r = 1$ and $\delta = 5$ which results in `GapFunction`s with dimension 1 and degree at most 5. To compute `GapFunction`s with a given dimension and degree, replace 1 in the first line with the desired dimension and replace 5 in the third line with the desired degree. For example to compute 3-dimensional `GapFunction`s with degree 5, this block must be run first for $r = 1$, then $r = 2$ and finally $r = 3$.

```
> r:= 1:
gapFunList:=GapList([]):
delta:=5;
dimension:=r;
for i from dimension-1 to delta do
 main(i,dimension, gapFunList);
od;
printf("-------------------------------------------------\n");
printf("Total number of %d-dimensional gap functions: %d\n",dimension,
    nops(gapFunList:-data[dimension]));
printf("-------------------------------------------------\n");
counter:=0:
previousDelta:=dimension-1:
for gapFun in gapFunList:-data[dimension] do
    currentDelta:=gapFun:-getDelta();
    if currentDelta<>previousDelta then
      previousDelta:=currentDelta;
      counter:=1;
    else
      counter:=counter+1;
    fi;
    printf("Table %d.%d.%d\n",gapFun:-getDelta(),gapFun:-getDimension(),counter);
    gapFun:-printGap();
    printf("\n");
```

```
od:
```

A.1 Maple implementation

```
> with(combinat):
```

```
> incrementList:=proc(index, finalValues, startValue::integer:=0)
 local listSize, i, indexList;
 indexList:=index;
 listSize:=nops(indexList);
 indexList[1]:=indexList[1]+1;
 for i from 1 to listSize do
  if indexList[i]>finalValues[i] and i<listSize then
   if finalValues[i]>=startValue then indexList[i]:=startValue else
      indexList[i]:=0; fi;
   indexList[i+1]:=indexList[i+1]+1;
  elif indexList[i]>finalValues[i] and i=listSize then
   if finalValues[i]>=startValue then indexList[i]:=startValue else
      indexList[i]:=0; fi;
   return indexList, false;
  else
   return indexList, true;
  fi;
 od;
end proc:

isOutOfBounds:=proc(currentList, finalList)
 local i;
 for i from 1 to nops(currentList) do
  if currentList[i]>finalList[i] then return true; fi;
 od;
 return false;
end proc:

getBoundary:=proc(currentList, finalList)
 local i;
 return [seq(min(currentList[i],finalList[i]),i=1..nops(currentList))];
end proc:
```

```
> unprotect('GapFunction', 'GapList');
module GapFunction()
 option object;

 local dimension::integer:=0; #holds the dimension of the gap function
```

```
local bounds::list; #bounds of the axes (a list)
local gapFun::Array:=Array(0..0); #the gap function is stored internally
    as n-dim Array

export ModuleApply::static := proc()
 Object(GapFunction, _passed);
end;

export ModuleCopy::static := proc(self::GapFunction, proto::GapFunction,
    sizeList::list,$)
 local indexList:=[];
 local i;
 for i from 1 to nops(sizeList) do
  indexList:=[op(indexList),0..sizeList[i]];
 od;
 self:-gapFun:=Array(op(indexList));
 self:-dimension:=nops(sizeList);
 self:-bounds:=sizeList;
end;

export ModulePrint::static := proc(self::GapFunction)
 #printArray(self:-gapFun);
 nprintf("GapFunction\ndim=%d\ndelta=%d",self:-dimension,self:-getDelta());
end;

export printGap := proc()
 printf("Dim=%d, delta=%d:\n\n",dimension,getDelta());
 printArray(gapFun);
end;

local printArray::static := proc(T::Array)
 local arraySize:=[upperbound(T)];
 local d:=nops(arraySize);
 local lx,ly,i,j;
 if d=0 then
  printf("0\n");
 elif d=1 then
  for i from 0 to upperbound(T) do
   printf("%d ",T[i]);
  od;
  printf("\n");
 elif d=2 then
  lx,ly:=upperbound(T);
  for j from ly by -1 to 0 do
   for i from 0 to lx do
    printf("%d ",T[i,j]);
   od;
```

```
   printf("\n");
  od;
 else
  local loopIndex:=[seq(0,i=1..d-2)];
  local finalValues:=arraySize[3..-1];
  local shouldContinue:=true;
  while shouldContinue do
    printf("Level %a=\n",loopIndex);
    lx,ly:=arraySize[1],arraySize[2];
    for j from ly by -1 to 0 do
     for i from 0 to lx do
       printf("%d ",T[i,j,op(loopIndex)]);
     od;
     printf("\n");
    od;
    printf("--------------\n");
    loopIndex,shouldContinue:=incrementList(loopIndex,finalValues);
  od;
  printf("\n");
 fi;

end;

export get:=proc(indexList)
 return gapFun[op(indexList)];
end;

export set:=proc(indexList, entry)
 gapFun[op(indexList)]:=entry;
end;

export getDelta:=proc()
 if dimension=0 then 0 else gapFun[op(bounds)] fi;
end;

export getDimension:=proc()
 return dimension;
end;

export getBounds:=proc()
 return bounds;
end;

export getAxis:=proc(axis::integer)
 local i;
 if axis<1 or axis>dimension then error "Invalid axis index: axis should
    be in the range 1..dimension."; fi;
```

```
 local index:=[seq(0,i=1..dimension)];
 local axisSize:=bounds[axis];
 local incrementVector:=[seq(0,i=1..dimension)];
 incrementVector[axis]:=1;

 index[axis]:=axisSize;
 local axisDelta:=get(index);
 if axisDelta=0 then return GapFunction([0]); fi;

 local deltaIndex:=axisSize;
 while get(index)=axisDelta and deltaIndex>0 do
  deltaIndex:=deltaIndex-1;
  index[axis]:=deltaIndex;
 od;

 if deltaIndex=0 then error "invalid gap function"; fi;

 deltaIndex:=deltaIndex+1;
 local axisGap:=GapFunction([deltaIndex]);
 for i from 1 to deltaIndex do
   index[axis]:=i;
   axisGap:-set([i],get(index));
 od;
 return axisGap;

end;

export getAxes:=proc()
 local i;
 local axisList:=[];
 for i from 1 to dimension do
  axisList:=[op(axisList),getAxis(i)];
 od;
 return axisList;
end;

export getHyperplane:=proc(hIndex::integer) #hIndex is the hyperplane
    index
 local gapIndex, tempSum;
 if hIndex<1 or hIndex>dimension then error "Invalid hyperplane index:
     should be in the range 1..dimension."; fi;
 local hSize:=[op(bounds[1..hIndex-1]),op(bounds[hIndex+1..-1])]; #hSize
     is the hyperplane size
 local lastPosition:=hSize;
 local lastPositionInGap:=bounds;
 lastPositionInGap[hIndex]:=0;
 local hDelta:=get(lastPositionInGap);
```

```
 local loopIndex:=[seq(0,1..dimension-1)];
 local shouldContinue:=true;
 local sumIndex:=add(lastPosition);
 while shouldContinue do
  gapIndex:=[op(loopIndex[1..hIndex-1]),0,op(loopIndex[hIndex..-1])];
  tempSum:=add(loopIndex);
  if get(gapIndex)=hDelta and sumIndex>tempSum then
   lastPosition:=loopIndex;
   sumIndex:=tempSum;
  fi;
  loopIndex,shouldContinue:=incrementList(loopIndex,hSize);
 od;

 local hyperplaneGap:=GapFunction(lastPosition);
 loopIndex:=[seq(0,1..dimension-1)];
 shouldContinue:=true;
 while shouldContinue do
  gapIndex:=[op(loopIndex[1..hIndex-1]),0,op(loopIndex[hIndex..-1])];
  hyperplaneGap:-set(loopIndex,get(gapIndex));
  loopIndex,shouldContinue:=incrementList(loopIndex,lastPosition);
 od;

 return hyperplaneGap;


end;

export isEqual:=proc(argGap::GapFunction)
 local i;
 #test dimension equality
 local argDimension:=argGap:-getDimension();
 if dimension<>argDimension then return false; fi;

 #test bounds equality
 local argBounds:=argGap:-getBounds();
 for i from 1 to dimension do
  if bounds[i]<>argBounds[i] then return false; fi;
 od;

 #test entries
 local loopIndex:=[seq(0,i=1..dimension)];
 local shouldContinue:=true;
 while shouldContinue do
  if get(loopIndex)<>argGap:-get(loopIndex) then return false; fi;
  loopIndex,shouldContinue:=incrementList(loopIndex, bounds);
 od;
 return true;
```

```
end;

export fillWith:=proc(gap::GapFunction, endingIndex::list)
 local d:=getDimension();
 local d_gap:=gap:-getDimension();
 if d<>d_gap then error "gap function dimesnions don't match"; fi;
 local bounds_gap:=gap:-getBounds();
 local i;
 for i from 1 to d do
  if bounds[i]<endingIndex[i] or bounds_gap[i]<endingIndex[i] then error
      "invalid ending index"; fi;
 od;
 local shouldContinue:=true;
 local loopIndex:= [seq(0,i=1..d)];
 while shouldContinue do
   set(loopIndex, gap:-get(loopIndex));
   loopIndex,shouldContinue:= incrementList(loopIndex,endingIndex);
 od;
 return;
end;

export makeCopy::static:=proc(gap::GapFunction)
 local finalIndex:=gap:-getBounds();
 local copyGap:=GapFunction(finalIndex);
 copyGap:-fillWith(gap, finalIndex);
 return copyGap;
end;

local permuteList:=proc(L::list, permutation::list)
 local i;
 if nops(L)<>nops(permutation) then error "list size does not match the
     permutation size"; fi;
 local listSize:=nops(L);
 local permutedList:=[seq(0,i=1..listSize)];
 for i from 1 to listSize do
  permutedList[i]:=L[permutation[i]];
 od;
 return permutedList;
end;

export permuteAxes:=proc(permutation::list)
 local i;
 if nops(permutation)<>dimension then error "the permutation size does
     not mach the dimension"; fi;

 local permutedBounds:=permuteList(bounds,permutation);
 local permutedGap:=GapFunction(permutedBounds);
```

```
  local shouldContinue:=true;
  local loopIndex:= [seq(0,i=1..dimension)];
  local permutedIndex;
  while shouldContinue do
    permutedIndex:=permuteList(loopIndex,permutation);
    permutedGap:-set(permutedIndex, get(loopIndex));
    loopIndex,shouldContinue:= incrementList(loopIndex,bounds);
  od;
  return permutedGap;
 end;

end module:

module GapList()
 option object;

 export data::list:=[]; #holds the list

 export ModuleApply::static := proc()
  Object(GapList, _passed);
 end;

 export ModuleCopy::static := proc(self::GapList, proto::GapList,
     initialList::list, $)
  self:-data:=initialList;
 end;

 export ModulePrint::static := proc(self::GapList)
  nprintf("GapList\n%a",self:-data);
 end;

 export get:=proc(index)
  return data[index];
 end;

end module:
```

```
> isSymmetry:=proc(T::GapFunction, gapList::list) local i, loopIndex,
    shouldContinue, gapFun, lx_gapFun, ly_gapFun, lz_gapFun,
    symmetry,dimension,gapFunSizeList, indexList,
    sizeList,permutationList, permutation;
   #isSymmetry returns true if T is a permutation of a gap function in
      gapList and returns false if it is a new gap function
   dimension:=T:-getDimension();
   permutationList:=permute(dimension);
   indexList:=[seq(0,i=1..dimension)];
```

```
  sizeList:=T:-getBounds();
  for gapFun in gapList do
    gapFunSizeList:=gapFun:-getBounds();
    for permutation in permutationList do
      symmetry:=true;
      for i from 1 to dimension while symmetry=true do
        if sizeList[permutation[i]]<>gapFunSizeList[i] then
            symmetry:=false; fi;
      od;
      loopIndex:=[seq(0,i=1..dimension)];
      shouldContinue:=true;
      while shouldContinue and symmetry=true do
        for i from 1 to dimension do
          indexList[permutation[i]]:=loopIndex[i];
        od;
        if T:-get(indexList)<>gapFun:-get(loopIndex) then symmetry:=false;
           fi;
        loopIndex,shouldContinue:=incrementList(loopIndex,gapFunSizeList);
      od;
      if symmetry=true then return true; fi;
    od;#end for permutation
  od;
  return false;
 end proc:
```

```
>
    fillGapFunction:=proc(lambda::GapFunction,delta::integer,startIndex::list,
    gapFunList::GapList) local shouldStop, gapFunctionSize,
    loopIndex,shouldContinue,
    numOfOnes,sumOfList,dimension,tempIndex,gamma,l,newProperty,skip,i,j,tempIndex2,tempInde
 shouldStop:=proc(currentIndex::list) local
     gapFun,newPropertyHolds,direction, i, previousEntryIndex;
  #first we check to see if the latest entry in the table satisfies the
      property that in one step the increment is at most 1
  local incrementPropertyHolds:=true;
  for i from 1 to nops(currentIndex) while incrementPropertyHolds do
   previousEntryIndex:=currentIndex;
   if currentIndex[i]>0 then
    previousEntryIndex[i]:=currentIndex[i]-1;
    if lambda:-get(currentIndex)-lambda:-get(previousEntryIndex)>1 then
        incrementPropertyHolds:=false fi;
   fi;
  od;
  if incrementPropertyHolds=false then return true fi; #in case the table
     is constructed badly we should stop and move on to the next table

  if lambda:-get(currentIndex)=delta then
```

```
 gapFun:=GapFunction(currentIndex);
 gapFun:-fillWith(lambda,currentIndex);
 #issymmetry,newproperty
 newPropertyHolds:=true;
 for i from 1 to gapFun:-getDimension() while newPropertyHolds do
  direction:=[seq(0,s=1..gapFun:-getDimension())];
  direction[i]:=1;
  newPropertyHolds:=newProperty(gapFun,direction);
 od;
 if newPropertyHolds and
     isSymmetry(gapFun,gapFunList:-data[gapFun:-getDimension()])=false
     then
  local d:=gapFun:-getDimension();
  gapFunList:-data[d]:=[op(gapFunList:-data[d]),gapFun];
 fi;
 return true;
else
 return false;
fi;
end proc:

numOfOnes:=proc(indexList::list) local index, i;
 index:=0;
 for i in indexList do
  if i=1 then index:=index+1; fi;
 od;
 return index;
end proc;

sumOfList:=proc(L::list) local s,i;
 s:=0;
 for i in L do
  s:=s+i;
 od;
 return s;
end proc;



newProperty:=proc(gapCandidate::GapFunction, direction); local
    semiGrouplist, i, j, index, elements, sum,
    finalPoint,point,dimension,isLessThanOrEqual;
 finalPoint:=gapCandidate:-getBounds()-direction;
 #if the gap function size is 1 in the axis of direction, there's
     nothing to check
 for  i in finalPoint do
  if i=0 then return true; fi;
```

```
 od;
 dimension:=gapCandidate:-getDimension();
 semiGrouplist:=[];
 shouldContinue:=true;
 loopIndex:=[seq(1,i=1..dimension)];

 while shouldContinue do
    if
        gapCandidate:-get(loopIndex)=gapCandidate:-get(loopIndex+direction)
        then
     semiGrouplist:=[op(semiGrouplist),loopIndex];
    fi;
    loopIndex,shouldContinue:=incrementList(loopIndex,finalPoint,1);
 od;
 #Computes the semigroup generated by semiGrouplist
 elements:=convert(semiGrouplist, set);
 i:=1;
 j:=1;
 while i<=nops(elements) do
   while j<=nops(elements) do
     sum:=elements[i]+elements[j];
     isLessThanOrEqual:=true;
     for index from 1 to dimension while isLessThanOrEqual do
       if sum[index]>finalPoint[index] then isLessThanOrEqual:=false; fi;

     od;

     if isLessThanOrEqual and member(sum,elements)=false then
       elements:=elements union {sum};
       i:=1;
       j:=1;
     fi;
     j:=j+1;
   od;
   i:=i+1;
   j:=i;
 od;
 for point in elements do
   if gapCandidate:-get(point)<>gapCandidate:-get(point+direction) then
      return false; fi;
 od;
 return true;
end proc:


if shouldStop(startIndex) then return; fi;
gapFunctionSize:=lambda:-getBounds();
```

```
dimension:=lambda:-getDimension();
loopIndex:=startIndex;
shouldContinue:=true;
loopIndex,shouldContinue:=incrementList(loopIndex,gapFunctionSize,1);
while shouldContinue do
 skip:=false;

#start upward propagation
if dimension>1 then
 for i from 1 to dimension while skip=false do
  for j from 1 to dimension while skip=false do
   if i<>j then
    if loopIndex[i]-1>=0 and loopIndex[j]-1>=0 then
     tempIndex:=loopIndex;
     tempIndex2:=loopIndex;
     tempIndex3:=loopIndex;
     tempIndex[i]:=tempIndex[i]-1;
     tempIndex[j]:=tempIndex[j]-1;
     tempIndex2[i]:=tempIndex2[i]-1;
     tempIndex3[j]:=tempIndex3[j]-1;
     if lambda:-get(tempIndex)<>lambda:-get(tempIndex2) then
      lambda:-set(loopIndex, lambda:-get(tempIndex3)+1);
      skip:=true;
     fi;
     if shouldStop(loopIndex) then return; fi;
    fi;
   fi;
  od;
 od;
 if skip=true then
  loopIndex,shouldContinue:=incrementList(loopIndex,gapFunctionSize,1);
  next;
 fi;
fi;
#start lemma 2.9
 if loopIndex[1]-1>=0 then
  tempIndex:=loopIndex;
  tempIndex[1]:=tempIndex[1]-1;
  gamma:=lambda:-get(tempIndex);
  l:=numOfOnes(loopIndex);
  if sumOfList(loopIndex)>2*gamma+2-l then
   lambda:-set(loopIndex,gamma);
   if shouldStop(loopIndex) then return; fi;
   #no need for next because lemma 2.9 is the last part of while
  else
   lambda:-set(loopIndex,gamma);
```

```
    fillGapFunction(GapFunction:-makeCopy(lambda),delta,loopIndex,
        gapFunList);
    lambda:-set(loopIndex,gamma+1);
    if shouldStop(loopIndex) then return; fi;
   fi;
  fi;



  loopIndex,shouldContinue:=incrementList(loopIndex,gapFunctionSize,1);
 od; #end of while
end proc:
```

```
> assembleGapFunction:=proc(axisGap::GapFunction,
   hyperplaneGap::GapFunction, delta)
 local num, dimension, Lx,Ly,i,j,gapFunList,lambda, XiPlane, hList,
    deltaBound,
    numberOfPlanes,planeListIndex,shouldContinue,shouldContinueInner,isCompatible,
    currentPlane, currentPlaneSize,currentPlaneDelta,hyperplaneGapIndex,
    hyperplaneFinalGapIndex, gapIndex, boundaryIndex, allOnes;
 dimension:=axisGap:-getDimension()+hyperplaneGap:-getDimension();
 gapFunList:=GapList([seq([],i=1..dimension)]);
 if dimension=2 then
  Lx:=2*(delta-1)+1;
  Ly:=2*(delta-1)+1;
  if axisGap:-getDelta()+hyperplaneGap:-getDelta()<=delta-1 then
      lambda:=GapFunction([Lx,Ly]);
      for i from 0 to Lx do
        if i>axisGap:-getBounds()[1] then
         lambda:-set([i,0],axisGap:-getDelta());
        else
         lambda:-set([i,0],axisGap:-get([i]));
        fi;
      od;
      for j from 0 to Ly do
        if j>hyperplaneGap:-getBounds()[1] then
         lambda:-set([0,j],hyperplaneGap:-getDelta());
        else
         lambda:-set([0,j],hyperplaneGap:-get([j]));
        fi;
      od;
      lambda:-set([1,1],1);
      fillGapFunction(lambda,delta,[1,1], gapFunList);
  fi;
 else
  hList:=[seq([],i=1..dimension)];
  hList[1]:=[hyperplaneGap];
```

```
for i from 2 to dimension do
 XiPlane:=hyperplaneGap:-getHyperplane(i-1);
 deltaBound:=delta-hyperplaneGap:-getAxis(i-1):-getDelta()-1;
 for j from 1 to deltaBound do
  hList[i]:=[op(hList[i]),op(assembleGapFunction(axisGap,XiPlane,j))];
 od;
od;
numberOfPlanes:=[seq(nops(hList[i]),i=1..dimension)];
for num in numberOfPlanes do
 if num=0 then return []; fi;
od;
planeListIndex:=[seq(1,i=1..dimension)];
shouldContinue:=true;
while shouldContinue do
 lambda:=GapFunction([seq(2*(delta-1)+1,i=1..dimension)]);
 isCompatible:=true;
 for i from 1 to dimension while isCompatible do
   currentPlane:=hList[i][planeListIndex[i]];
   currentPlaneSize:=currentPlane:-getBounds();
   currentPlaneDelta:=currentPlane:-getDelta();
   hyperplaneGapIndex:=[seq(0,i=1..dimension-1)];
   hyperplaneFinalGapIndex:=[seq(2*(delta-1)+1,i=1..dimension-1)];
   shouldContinueInner:=true;
   while shouldContinueInner do
    if i<dimension then
     gapIndex:=[op(hyperplaneGapIndex[1..i-1]),0,op(hyperplaneGapIndex[i..-1])];
    else
     gapIndex:=[op(hyperplaneGapIndex[1..i-1]),0];
    fi;
    if isOutOfBounds(hyperplaneGapIndex, currentPlaneSize) then
     boundaryIndex:=getBoundary(hyperplaneGapIndex, currentPlaneSize);
     for j from 1 to i-1 while isCompatible do
      if gapIndex[j]=0 and
          lambda:-get(gapIndex)<>currentPlane:-get(boundaryIndex) then
          isCompatible:=false; fi;
     od;
     lambda:-set(gapIndex, currentPlane:-get(boundaryIndex));
    else
     for j from 1 to i-1 while isCompatible do
      if gapIndex[j]=0 and
          lambda:-get(gapIndex)<>currentPlane:-get(hyperplaneGapIndex)
          then isCompatible:=false; fi;
     od;
     lambda:-set(gapIndex, currentPlane:-get(hyperplaneGapIndex));
    fi;
    hyperplaneGapIndex,shouldContinueInner:=incrementList(hyperplaneGapIndex,hyperplaneFi
   od; #end while shouldContinueInner
```

```
     #improve the runtime? if isCompatible=false then
         planeListIndex[i]:=planeListIndex[i]+1; fi;
    od; #end for i

    if isCompatible then
     allOnes:=[seq(1,i=1..dimension)];
     lambda:-set(allOnes, dimension-1);
     fillGapFunction(lambda,delta,allOnes,gapFunList);
    fi;
    planeListIndex,shouldContinue:=incrementList(planeListIndex,numberOfPlanes,1);
   od;
  fi;
  return gapFunList:-data[dimension];
end proc:
```

```
> main:=proc(delta,dimension, gapFunList::GapList) local Lx, Ly, xAxis,
    yAxis, i,j, k, m, lambda,assembled, gapFun, permutationList, axisGap,
    hyperplaneGap, hyperplaneIndex, axis, hyperplane;

 #check to see if the number of entris in gapFunList equals dimension, if
    not add empty lists to it until it becomes so
 if nops(gapFunList:-data)<dimension then
  for i from nops(gapFunList:-data)+1 to dimension do
    gapFunList:-data:=[op(gapFunList:-data),[]];
  od;
 fi;

 if dimension=1 then
  if delta=0 then
    lambda:=GapFunction([0]);
    gapFunList:-data[1]:=[op(gapFunList:-data[1]),lambda];
    return;
  fi;
  Lx:=2*(delta-1)+2;
  lambda:=GapFunction([Lx]);
  lambda:-set([1],0);
  fillGapFunction(lambda,delta,[1], gapFunList);
 elif dimension=2 then
   for k from 1 to nops(gapFunList:-data[1]) do
    for m from k to nops(gapFunList:-data[1]) do
      xAxis:=gapFunList:-data[1][m];
      yAxis:=gapFunList:-data[1][k];
      if xAxis:-getDelta()+yAxis:-getDelta()<=delta-1 then
       assembled:=assembleGapFunction(xAxis, yAxis,delta);
       for gapFun in assembled do
        if isSymmetry(gapFun, gapFunList:-data[dimension])= false then
```

```
        gapFunList:-data[dimension]:=[op(gapFunList:-data[dimension]),
            gapFun];
       fi
      od;
     fi;
   od; #end for m
  od; #end for k
 else #dimension>2
  for k from 1 to nops(gapFunList:-data[1]) do
   for m from 1 to nops(gapFunList:-data[dimension-1]) do
     axis:=gapFunList:-data[1][k];
     hyperplane:=gapFunList:-data[dimension-1][m];
     if axis:-getDelta()+hyperplane:-getDelta()<=delta-1 then
      assembled:=assembleGapFunction(axis, hyperplane,delta);
      for gapFun in assembled do
       if isSymmetry(gapFun, gapFunList:-data[dimension])= false then
        gapFunList:-data[dimension]:=[op(gapFunList:-data[dimension]),
            gapFun];
       fi
      od;
     fi;
   od; #end for m
  od; #end for k
 fi; #end if dimension
end proc:
```

Appendix B

Ouput of algorithm for degree 5

In this appendix we include the output of our algorithm for $\delta = 5$. These tables are `GapFunction`s with degree 5 containing all gap functions with degree 5. There may or may not be tables that are not valid gap functions. We have separated the output based on the dimension r of the tables.

B.1 `GapFunctions` with $r = 1$

```
                        delta := 5
                      dimension := 1
-------------------------------------------------
Total number of 1-dimensional gap functions: 12
-------------------------------------------------
Table 5.1.1
Dim=1, delta=5:
0 0 1 1 2 2 3 3 4 4 5
Table 5.1.2
Dim=1, delta=5:
0 0 1 2 2 3 4 4 4 5
Table 5.1.3
Dim=1, delta=5:
0 0 1 2 2 3 4 4 5
Table 5.1.4
Dim=1, delta=5:
0 0 1 2 3 3 3 4 5
Table 5.1.5
Dim=1, delta=5:
0 0 1 2 3 3 4 4 4 4 5
Table 5.1.6
Dim=1, delta=5:
0 0 1 2 3 3 4 4 5
Table 5.1.7
```

```
Dim=1, delta=5:
0 0 1 2 3 3 4 5
Table 5.1.8
Dim=1, delta=5:
0 0 1 2 3 4 4 4 4 5
Table 5.1.9
Dim=1, delta=5:
0 0 1 2 3 4 4 4 4 5
Table 5.1.10
Dim=1, delta=5:
0 0 1 2 3 4 4 4 5
Table 5.1.11
Dim=1, delta=5:
0 0 1 2 3 4 4 5
Table 5.1.12
Dim=1, delta=5:
0 0 1 2 3 4 5
```

B.2 GapFunctions with $r = 2$

```
                        delta := 5
                      dimension := 2
-------------------------------------------------
Total number of 2-dimensional gap functions: 42
-------------------------------------------------
Table 5.2.1
Dim=2, delta=5:
0 1 2 3 4 5
0 1 2 3 4 4
0 1 2 3 3 3
0 1 2 2 2 2
0 1 1 1 1 1
0 0 0 0 0 0
Table 5.2.2
Dim=2, delta=5:
0 1 2 3 4 4 5
0 1 2 3 4 4 4
0 1 2 2 3 3 3
0 0 1 1 2 2 2
Table 5.2.3
Dim=2, delta=5:
0 1 2 3 4 5
0 1 2 3 4 4
0 1 2 2 3 3
0 0 1 1 2 2
```

Table 5.2.4
Dim=2, delta=5:
0 1 2 3 4 4 4 5
0 1 2 3 4 4 4 4
0 1 2 3 3 3 3 3
0 0 1 2 2 2 2 2
Table 5.2.5
Dim=2, delta=5:
0 1 2 2 3 3 4 4 5
0 1 2 2 3 3 4 4 4
0 0 1 1 2 2 3 3 3
Table 5.2.6
Dim=2, delta=5:
0 1 2 3 4 4 5
0 1 2 2 3 3 4
0 0 1 1 2 2 3
Table 5.2.7
Dim=2, delta=5:
0 1 2 3 3 4 5
0 1 2 3 3 3 4
0 0 1 2 2 2 3
Table 5.2.8
Dim=2, delta=5:
0 1 2 3 4 4 5
0 1 2 3 3 3 4
0 0 1 2 2 2 3
Table 5.2.9
Dim=2, delta=5:
0 1 2 3 3 4 4 4 5
0 1 2 3 3 4 4 4 4
0 0 1 2 2 3 3 3 3
Table 5.2.10
Dim=2, delta=5:
0 1 2 3 3 4 5
0 1 2 3 3 4 4
0 0 1 2 2 3 3
Table 5.2.11
Dim=2, delta=5:
0 1 2 3 4 5
0 1 2 3 3 4
0 0 1 2 2 3
Table 5.2.12
Dim=2, delta=5:
0 1 2 3 4 4 4 4 5
0 1 2 3 4 4 4 4 4
0 0 1 2 3 3 3 3 3
Table 5.2.13

Dim=2, delta=5:
0 1 2 3 4 4 4 5
0 1 2 3 4 4 4 4
0 0 1 2 3 3 3 3
Table 5.2.14
Dim=2, delta=5:
0 1 2 3 4 4 5
0 1 2 3 4 4 4
0 0 1 2 3 3 3
Table 5.2.15
Dim=2, delta=5:
0 1 2 3 4 5
0 1 2 3 4 4
0 0 1 2 3 3
Table 5.2.16
Dim=2, delta=5:
0 1 2 2 3 3 4 4 5
0 0 1 1 2 2 3 3 4
Table 5.2.17
Dim=2, delta=5:
0 1 2 3 3 4 4 4 5
0 0 1 2 2 3 3 3 4
Table 5.2.18
Dim=2, delta=5:
0 1 2 3 3 4 5
0 0 1 2 2 3 4
Table 5.2.19
Dim=2, delta=5:
0 1 2 3 4 4 4 4 5
0 0 1 2 3 3 3 3 4
Table 5.2.20
Dim=2, delta=5:
0 1 2 3 4 4 4 5
0 0 1 2 3 3 3 4
Table 5.2.21
Dim=2, delta=5:
0 1 2 3 4 4 5
0 0 1 2 3 3 4
Table 5.2.22
Dim=2, delta=5:
0 1 2 3 4 5
0 0 1 2 3 4
Table 5.2.23
Dim=2, delta=5:
1 2 3 4 5
1 2 3 3 4
1 2 3 3 3

0 1 2 2 2
0 0 1 1 1
Table 5.2.24
Dim=2, delta=5:
1 2 3 4 4 5
1 2 3 4 4 4
1 2 3 4 4 4
1 2 3 3 3 3
0 1 2 2 2 2
0 0 1 1 1 1
Table 5.2.25
Dim=2, delta=5:
1 2 3 4 5
1 2 3 4 4
1 2 3 3 3
0 1 2 2 2
0 0 1 1 1
Table 5.2.26
Dim=2, delta=5:
1 2 3 3 4 4 5
1 2 3 3 4 4 4
1 2 3 3 4 4 4
0 1 2 2 3 3 3
0 0 1 1 2 2 2
Table 5.2.27
Dim=2, delta=5:
1 2 3 4 5
1 2 3 3 4
1 2 3 3 4
0 1 2 2 3
0 0 1 1 2
Table 5.2.28
Dim=2, delta=5:
1 2 3 3 4 4 5
1 2 3 3 4 4 4
0 1 2 2 3 3 3
0 0 1 1 2 2 2
Table 5.2.29
Dim=2, delta=5:
1 2 3 4 5
1 2 3 3 4
0 1 2 2 3
0 0 1 1 2
Table 5.2.30
Dim=2, delta=5:
1 2 3 4 4 4 5
1 2 3 4 4 4 4

```
1 2 3 4 4 4 4
0 1 2 3 3 3 3
0 0 1 2 2 2 2
Table 5.2.31
Dim=2, delta=5:
1 2 3 4 4 5
1 2 3 4 4 4
1 2 3 4 4 4
0 1 2 3 3 3
0 0 1 2 2 2
Table 5.2.32
Dim=2, delta=5:
1 2 3 4 5
1 2 3 4 4
1 2 3 4 4
0 1 2 3 3
0 0 1 2 2
Table 5.2.33
Dim=2, delta=5:
1 2 3 4 4 4 5
1 2 3 4 4 4 4
0 1 2 3 3 3 3
0 0 1 2 2 2 2
Table 5.2.34
Dim=2, delta=5:
1 2 3 4 4 5
1 2 3 4 4 4
0 1 2 3 3 3
0 0 1 2 2 2
Table 5.2.35
Dim=2, delta=5:
1 2 3 4 5
1 2 3 4 4
0 1 2 3 3
0 0 1 2 2
Table 5.2.36
Dim=2, delta=5:
1 2 3 3 4 4 5
0 1 2 2 3 3 4
0 0 1 1 2 2 3
Table 5.2.37
Dim=2, delta=5:
1 2 3 4 4 4 5
0 1 2 3 3 3 4
0 0 1 2 2 2 3
Table 5.2.38
Dim=2, delta=5:
```

```
1 2 3 4 4 5
0 1 2 3 3 4
0 0 1 2 2 3
Table 5.2.39
Dim=2, delta=5:
1 2 3 4 5
0 1 2 3 4
0 0 1 2 3
Table 5.2.40
Dim=2, delta=5:
2 3 4 4 5
1 2 3 3 4
1 2 3 3 4
0 1 2 2 3
0 0 1 1 2
Table 5.2.41
Dim=2, delta=5:
2 3 4 5
1 2 3 4
1 2 3 4
0 1 2 3
0 0 1 2
Table 5.2.42
Dim=2, delta=5:
2 3 4 5
1 2 3 4
0 1 2 3
0 0 1 2
```

B.3 GapFunctions with $r = 3$

```
                        delta := 5
                      dimension := 3
-------------------------------------------------
Total number of 3-dimensional gap functions: 42
-------------------------------------------------
Table 5.3.1
Dim=3, delta=5:
Level [0]=
0 1 1 1 1
0 1 1 1 1
0 0 0 0 0
--------------
Level [1]=
1 2 2 2 2
```

```
1 2 2 2 2
0 1 1 1 1
--------------
Level [2]=
1 2 3 3 3
1 2 3 3 3
0 1 2 2 2
--------------
Level [3]=
1 2 3 4 4
1 2 3 4 4
0 1 2 3 3
--------------
Level [4]=
1 2 3 4 5
1 2 3 4 4
0 1 2 3 3
--------------
Table 5.3.2
Dim=3, delta=5:
Level [0]=
0 1 1 1 1
0 0 0 0 0
--------------
Level [1]=
1 2 2 2 2
0 1 1 1 1
--------------
Level [2]=
1 2 3 3 3
0 1 2 2 2
--------------
Level [3]=
1 2 3 4 4
0 1 2 3 3
--------------
Level [4]=
1 2 3 4 5
0 1 2 3 4
--------------
Table 5.3.3
Dim=3, delta=5:
Level [0]=
0 1 2 2
0 1 2 2
0 1 1 1
0 0 0 0
```

```
--------------
Level [1]=
1 2 3 3
1 2 3 3
1 2 2 2
0 1 1 1
--------------
Level [2]=
2 3 4 4
2 3 4 4
1 2 3 3
0 1 2 2
--------------
Level [3]=
2 3 4 5
2 3 4 4
1 2 3 3
0 1 2 2
--------------
Table 5.3.4
Dim=3, delta=5:
Level [0]=
0 1 2 2
0 1 1 1
0 0 0 0
--------------
Level [1]=
1 2 3 3
1 2 2 2
0 1 1 1
--------------
Level [2]=
2 3 4 4
1 2 3 3
0 1 2 2
--------------
Level [3]=
2 3 4 5
1 2 3 4
0 1 2 3
--------------
Table 5.3.5
Dim=3, delta=5:
Level [0]=
1 2 2 2
1 2 2 2
1 2 2 2
```

```
0 1 1 1
0 0 0 0
--------------
Level [1]=
2 3 3 3
2 3 3 3
2 3 3 3
1 2 2 2
0 1 1 1
--------------
Level [2]=
2 3 4 4
2 3 4 4
2 3 4 4
1 2 3 3
0 1 2 2
--------------
Level [3]=
2 3 4 5
2 3 4 4
2 3 4 4
1 2 3 3
0 1 2 2
--------------
Table 5.3.6
Dim=3, delta=5:
Level [0]=
1 2 2 2
1 2 2 2
0 1 1 1
0 0 0 0
--------------
Level [1]=
2 3 3 3
2 3 3 3
1 2 2 2
0 1 1 1
--------------
Level [2]=
2 3 4 4
2 3 4 4
1 2 3 3
0 1 2 2
--------------
Level [3]=
2 3 4 5
2 3 4 4
```

```
1 2 3 3
0 1 2 2
--------------
Table 5.3.7
Dim=3, delta=5:
Level [0]=
1 2 2 2
0 1 1 1
0 0 0 0
--------------
Level [1]=
2 3 3 3
1 2 2 2
0 1 1 1
--------------
Level [2]=
2 3 4 4
1 2 3 3
0 1 2 2
--------------
Level [3]=
2 3 4 5
1 2 3 4
0 1 2 3
--------------
Table 5.3.8
Dim=3, delta=5:
Level [0]=
1 2 3 3
1 2 3 3
1 2 3 3
1 2 2 2
0 1 1 1
0 0 0 0
--------------
Level [1]=
2 3 4 4
2 3 4 4
2 3 4 4
2 3 3 3
1 2 2 2
0 1 1 1
--------------
Level [2]=
2 3 4 5
2 3 4 4
2 3 4 4
```

```
2 3 3 3
1 2 2 2
0 1 1 1
--------------
Table 5.3.9
Dim=3, delta=5:
Level [0]=
1 2 3 4
1 2 3 3
1 2 3 3
1 2 2 2
0 1 1 1
0 0 0 0
--------------
Level [1]=
2 3 4 5
2 3 4 4
2 3 4 4
2 3 3 3
1 2 2 2
0 1 1 1
--------------
Table 5.3.10
Dim=3, delta=5:
Level [0]=
1 2 3
1 2 2
1 2 2
0 1 1
0 0 0
--------------
Level [1]=
2 3 4
2 3 3
2 3 3
1 2 2
0 1 1
--------------
Level [2]=
3 4 5
2 3 4
2 3 3
1 2 2
0 1 1
--------------
Table 5.3.11
Dim=3, delta=5:
```

```
Level [0]=
1 2 3
1 2 2
1 2 2
0 1 1
0 0 0
--------------
Level [1]=
2 3 4
2 3 3
2 3 3
1 2 2
0 1 1
--------------
Level [2]=
3 4 5
2 3 4
2 3 4
1 2 3
0 1 2
--------------
Table 5.3.12
Dim=3, delta=5:
Level [0]=
1 2 3
1 2 3
1 2 2
0 1 1
0 0 0
--------------
Level [1]=
2 3 4
2 3 4
2 3 3
1 2 2
0 1 1
--------------
Level [2]=
3 4 5
2 3 4
2 3 3
1 2 2
0 1 1
--------------
Table 5.3.13
Dim=3, delta=5:
Level [0]=
```

```
1 2 3
1 2 3
1 2 2
0 1 1
0 0 0
--------------
Level [1]=
2 3 4
2 3 4
2 3 3
1 2 2
0 1 1
--------------
Level [2]=
3 4 5
3 4 4
2 3 3
1 2 2
0 1 1
--------------
Table 5.3.14
Dim=3, delta=5:
Level [0]=
1 2 3
1 2 2
0 1 1
0 0 0
--------------
Level [1]=
2 3 4
2 3 3
1 2 2
0 1 1
--------------
Level [2]=
3 4 5
2 3 4
1 2 3
0 1 2
--------------
Table 5.3.15
Dim=3, delta=5:
Level [0]=
2 3 3
2 3 3
2 3 3
1 2 2
```

```
1 2 2
0 1 1
0 0 0
--------------
Level [1]=
3 4 4
3 4 4
3 4 4
2 3 3
2 3 3
1 2 2
0 1 1
--------------
Level [2]=
3 4 5
3 4 4
3 4 4
2 3 3
2 3 3
1 2 2
0 1 1
--------------
Table 5.3.16
Dim=3, delta=5:
Level [0]=
2 3 3
1 2 2
1 2 2
0 1 1
0 0 0
--------------
Level [1]=
3 4 4
2 3 3
2 3 3
1 2 2
0 1 1
--------------
Level [2]=
3 4 5
2 3 4
2 3 3
1 2 2
0 1 1
--------------
Table 5.3.17
Dim=3, delta=5:
```

```
Level [0]=
2 3 3
1 2 2
1 2 2
0 1 1
0 0 0
--------------
Level [1]=
3 4 4
2 3 3
2 3 3
1 2 2
0 1 1
--------------
Level [2]=
3 4 5
2 3 4
2 3 4
1 2 3
0 1 2
--------------
Table 5.3.18
Dim=3, delta=5:
Level [0]=
2 3 4
2 3 3
2 3 3
1 2 2
1 2 2
0 1 1
0 0 0
--------------
Level [1]=
3 4 5
3 4 4
3 4 4
2 3 3
2 3 3
1 2 2
0 1 1
--------------
Table 5.3.19
Dim=3, delta=5:
Level [0]=
2 3 4
1 2 3
1 2 2
```

```
0 1 1
0 0 0
--------------
Level [1]=
3 4 5
2 3 4
2 3 3
1 2 2
0 1 1
--------------
Table 5.3.20
Dim=3, delta=5:
Level [0]=
2 3 3
2 3 3
2 3 3
2 3 3
1 2 2
0 1 1
0 0 0
--------------
Level [1]=
3 4 4
3 4 4
3 4 4
3 4 4
2 3 3
1 2 2
0 1 1
--------------
Level [2]=
3 4 5
3 4 4
3 4 4
3 4 4
2 3 3
1 2 2
0 1 1
--------------
Table 5.3.21
Dim=3, delta=5:
Level [0]=
2 3 3
2 3 3
2 3 3
1 2 2
0 1 1
```

```
0 0 0
--------------
Level [1]=
3 4 4
3 4 4
3 4 4
2 3 3
1 2 2
0 1 1
--------------
Level [2]=
3 4 5
3 4 4
3 4 4
2 3 3
1 2 2
0 1 1
--------------
Table 5.3.22
Dim=3, delta=5:
Level [0]=
2 3 3
2 3 3
1 2 2
0 1 1
0 0 0
--------------
Level [1]=
3 4 4
3 4 4
2 3 3
1 2 2
0 1 1
--------------
Level [2]=
3 4 5
3 4 4
2 3 3
1 2 2
0 1 1
--------------
Table 5.3.23
Dim=3, delta=5:
Level [0]=
2 3 3
1 2 2
0 1 1
```

```
0 0 0
--------------
Level [1]=
3 4 4
2 3 3
1 2 2
0 1 1
--------------
Level [2]=
3 4 5
2 3 4
1 2 3
0 1 2
--------------
Table 5.3.24
Dim=3, delta=5:
Level [0]=
2 3 4
2 3 3
2 3 3
2 3 3
1 2 2
0 1 1
0 0 0
--------------
Level [1]=
3 4 5
3 4 4
3 4 4
3 4 4
2 3 3
1 2 2
0 1 1
--------------
Table 5.3.25
Dim=3, delta=5:
Level [0]=
2 3 4
2 3 3
2 3 3
1 2 2
0 1 1
0 0 0
--------------
Level [1]=
3 4 5
3 4 4
```

```
3 4 4
2 3 3
1 2 2
0 1 1
--------------
Table 5.3.26
Dim=3, delta=5:
Level [0]=
2 3 4
2 3 3
1 2 2
0 1 1
0 0 0
--------------
Level [1]=
3 4 5
3 4 4
2 3 3
1 2 2
0 1 1
--------------
Table 5.3.27
Dim=3, delta=5:
Level [0]=
1 2 2
1 2 2
1 2 2
0 1 1
0 0 0
--------------
Level [1]=
2 3 3
2 3 3
2 3 3
1 2 2
0 1 1
--------------
Level [2]=
3 4 4
3 4 4
3 4 4
2 3 3
1 2 2
--------------
Level [3]=
3 4 4
3 4 4
```

```
3 4 4
2 3 3
1 2 2
--------------
Level [4]=
3 4 5
3 4 4
3 4 4
2 3 3
1 2 2
--------------
Table 5.3.28
Dim=3, delta=5:
Level [0]=
1 2 2
1 2 2
0 1 1
0 0 0
--------------
Level [1]=
2 3 3
2 3 3
1 2 2
0 1 1
--------------
Level [2]=
3 4 4
3 4 4
2 3 3
1 2 2
--------------
Level [3]=
3 4 4
3 4 4
2 3 3
1 2 2
--------------
Level [4]=
3 4 5
3 4 4
2 3 3
1 2 2
--------------
Table 5.3.29
Dim=3, delta=5:
Level [0]=
1 2 2
```

```
1 2 2
0 1 1
0 0 0
--------------
Level [1]=
2 3 3
2 3 3
1 2 2
0 1 1
--------------
Level [2]=
3 4 4
3 4 4
2 3 3
1 2 2
--------------
Level [3]=
3 4 5
3 4 4
2 3 3
1 2 2
--------------
Table 5.3.30
Dim=3, delta=5:
Level [0]=
1 2 2
0 1 1
0 0 0
--------------
Level [1]=
2 3 3
1 2 2
0 1 1
--------------
Level [2]=
3 4 4
2 3 3
1 2 2
--------------
Level [3]=
3 4 4
2 3 3
1 2 2
--------------
Level [4]=
3 4 5
2 3 4
```

```
1 2 3
--------------
Table 5.3.31
Dim=3, delta=5:
Level [0]=
1 2 2
0 1 1
0 0 0
--------------
Level [1]=
2 3 3
1 2 2
0 1 1
--------------
Level [2]=
3 4 4
2 3 3
1 2 2
--------------
Level [3]=
3 4 5
2 3 4
1 2 3
--------------
Table 5.3.32
Dim=3, delta=5:
Level [0]=
3 4
2 3
2 3
1 2
1 2
0 1
0 0
--------------
Level [1]=
4 5
3 4
3 4
2 3
2 3
1 2
0 1
--------------
Table 5.3.33
Dim=3, delta=5:
Level [0]=
```

```
3 4
2 3
2 3
2 3
1 2
0 1
0 0
--------------
Level [1]=
4 5
3 4
3 4
3 4
2 3
1 2
0 1
--------------
Table 5.3.34
Dim=3, delta=5:
Level [0]=
3 4
2 3
2 3
1 2
0 1
0 0
--------------
Level [1]=
4 5
3 4
3 4
2 3
1 2
0 1
--------------
Table 5.3.35
Dim=3, delta=5:
Level [0]=
3 4
2 3
1 2
0 1
0 0
--------------
Level [1]=
4 5
3 4
```

```
2 3
1 2
0 1
--------------
Table 5.3.36
Dim=3, delta=5:
Level [0]=
1 2
1 2
1 2
0 1
0 0
--------------
Level [1]=
2 3
2 3
2 3
1 2
0 1
--------------
Level [2]=
3 4
3 4
3 4
2 3
1 2
--------------
Level [3]=
3 4
3 4
3 4
2 3
1 2
--------------
Level [4]=
4 5
3 4
3 4
2 3
1 2
--------------
Table 5.3.37
Dim=3, delta=5:
Level [0]=
1 2
1 2
0 1
```

```
0 0
--------------
Level [1]=
2 3
2 3
1 2
0 1
--------------
Level [2]=
3 4
3 4
2 3
1 2
--------------
Level [3]=
3 4
3 4
2 3
1 2
--------------
Level [4]=
4 5
3 4
2 3
1 2
--------------
Table 5.3.38
Dim=3, delta=5:
Level [0]=
1 2
1 2
0 1
0 0
--------------
Level [1]=
2 3
2 3
1 2
0 1
--------------
Level [2]=
3 4
3 4
2 3
1 2
--------------
Level [3]=
```

```
4 5
3 4
2 3
1 2
--------------
Table 5.3.39
Dim=3, delta=5:
Level [0]=
2 3
1 2
1 2
0 1
0 0
--------------
Level [1]=
3 4
2 3
2 3
1 2
0 1
--------------
Level [2]=
4 5
3 4
3 4
2 3
1 2
--------------
Table 5.3.40
Dim=3, delta=5:
Level [0]=
2 3
1 2
0 1
0 0
--------------
Level [1]=
3 4
2 3
1 2
0 1
--------------
Level [2]=
4 5
3 4
2 3
1 2
```

```
--------------
Table 5.3.41
Dim=3, delta=5:
Level [0]=
0 1 2 3 3
0 1 2 2 2
0 0 1 1 1
--------------
Level [1]=
1 2 3 4 4
1 2 3 3 3
0 1 2 2 2
--------------
Level [2]=
2 3 4 4 5
1 2 3 3 4
0 1 2 2 3
--------------
Table 5.3.42
Dim=3, delta=5:
Level [0]=
1 2 3
0 1 2
0 0 1
--------------
Level [1]=
2 3 4
1 2 3
0 1 2
--------------
Level [2]=
3 4 5
2 3 4
1 2 3
```

B.4 GapFunctions with $r = 4$

```
                        delta := 5
                      dimension := 4
-------------------------------------------------
Total number of 4-dimensional gap functions: 17
-------------------------------------------------
Table 5.4.1
Dim=4, delta=5:
Level [0, 0]=
```

```
0 1 1 1
0 1 1 1
0 0 0 0
--------------
Level [1, 0]=
1 2 2 2
1 2 2 2
0 1 1 1
--------------
Level [2, 0]=
1 2 2 2
1 2 2 2
0 1 1 1
--------------
Level [0, 1]=
1 2 2 2
1 2 2 2
0 1 1 1
--------------
Level [1, 1]=
2 3 3 3
2 3 3 3
1 2 2 2
--------------
Level [2, 1]=
2 3 3 3
2 3 3 3
1 2 2 2
--------------
Level [0, 2]=
1 2 3 3
1 2 3 3
0 1 2 2
--------------
Level [1, 2]=
2 3 4 4
2 3 4 4
1 2 3 3
--------------
Level [2, 2]=
2 3 4 4
2 3 4 4
1 2 3 3
--------------
Level [0, 3]=
1 2 3 3
1 2 3 3
```

```
0 1 2 2
--------------
Level [1, 3]=
2 3 4 4
2 3 4 4
1 2 3 3
--------------
Level [2, 3]=
2 3 4 5
2 3 4 4
1 2 3 3
--------------
Table 5.4.2
Dim=4, delta=5:
Level [0, 0]=
0 1 1 1
0 0 0 0
--------------
Level [1, 0]=
1 2 2 2
0 1 1 1
--------------
Level [2, 0]=
1 2 2 2
0 1 1 1
--------------
Level [0, 1]=
1 2 2 2
0 1 1 1
--------------
Level [1, 1]=
2 3 3 3
1 2 2 2
--------------
Level [2, 1]=
2 3 3 3
1 2 2 2
--------------
Level [0, 2]=
1 2 3 3
0 1 2 2
--------------
Level [1, 2]=
2 3 4 4
1 2 3 3
--------------
Level [2, 2]=
```

```
2 3 4 4
1 2 3 3
--------------
Level [0, 3]=
1 2 3 3
0 1 2 2
--------------
Level [1, 3]=
2 3 4 4
1 2 3 3
--------------
Level [2, 3]=
2 3 4 5
1 2 3 4
--------------
Table 5.4.3
Dim=4, delta=5:
Level [0, 0]=
0 1 1 1
0 0 0 0
--------------
Level [1, 0]=
1 2 2 2
0 1 1 1
--------------
Level [0, 1]=
1 2 2 2
0 1 1 1
--------------
Level [1, 1]=
2 3 3 3
1 2 2 2
--------------
Level [0, 2]=
1 2 3 3
0 1 2 2
--------------
Level [1, 2]=
2 3 4 4
1 2 3 3
--------------
Level [0, 3]=
1 2 3 4
0 1 2 3
--------------
Level [1, 3]=
2 3 4 5
```

```
1 2 3 4
--------------
Table 5.4.4
Dim=4, delta=5:
Level [0, 0]=
0 1 1
0 1 1
0 0 0
--------------
Level [1, 0]=
1 2 2
1 2 2
0 1 1
--------------
Level [2, 0]=
1 2 3
1 2 2
0 1 1
--------------
Level [0, 1]=
1 2 2
1 2 2
0 1 1
--------------
Level [1, 1]=
2 3 3
2 3 3
1 2 2
--------------
Level [2, 1]=
2 3 4
2 3 3
1 2 2
--------------
Level [0, 2]=
1 2 3
1 2 2
0 1 1
--------------
Level [1, 2]=
2 3 4
2 3 3
1 2 2
--------------
Level [2, 2]=
3 4 5
2 3 4
```

```
1 2 3
--------------
Table 5.4.5
Dim=4, delta=5:
Level [0, 0]=
0 1 1
0 1 1
0 0 0
--------------
Level [1, 0]=
1 2 2
1 2 2
0 1 1
--------------
Level [2, 0]=
1 2 3
1 2 2
0 1 1
--------------
Level [0, 1]=
1 2 2
1 2 2
0 1 1
--------------
Level [1, 1]=
2 3 3
2 3 3
1 2 2
--------------
Level [2, 1]=
2 3 4
2 3 3
1 2 2
--------------
Level [0, 2]=
1 2 3
1 2 3
0 1 2
--------------
Level [1, 2]=
2 3 4
2 3 4
1 2 3
--------------
Level [2, 2]=
3 4 5
2 3 4
```

```
1 2 3
--------------
Table 5.4.6
Dim=4, delta=5:
Level [0, 0]=
0 1 1
0 1 1
0 0 0
--------------
Level [1, 0]=
1 2 2
1 2 2
0 1 1
--------------
Level [2, 0]=
2 3 3
2 3 3
1 2 2
--------------
Level [3, 0]=
2 3 3
2 3 3
1 2 2
--------------
Level [4, 0]=
2 3 3
2 3 3
1 2 2
--------------
Level [0, 1]=
1 2 2
1 2 2
0 1 1
--------------
Level [1, 1]=
2 3 3
2 3 3
1 2 2
--------------
Level [2, 1]=
3 4 4
3 4 4
2 3 3
--------------
Level [3, 1]=
3 4 4
3 4 4
```

```
2 3 3
--------------
Level [4, 1]=
3 4 4
3 4 4
2 3 3
--------------
Level [0, 2]=
1 2 2
1 2 2
0 1 1
--------------
Level [1, 2]=
2 3 3
2 3 3
1 2 2
--------------
Level [2, 2]=
3 4 4
3 4 4
2 3 3
--------------
Level [3, 2]=
3 4 4
3 4 4
2 3 3
--------------
Level [4, 2]=
3 4 5
3 4 4
2 3 3
--------------
Table 5.4.7
Dim=4, delta=5:
Level [0, 0]=
0 1 1
0 1 1
0 0 0
--------------
Level [1, 0]=
1 2 2
1 2 2
0 1 1
--------------
Level [2, 0]=
2 3 3
2 3 3
```

```
1 2 2
--------------
Level [3, 0]=
2 3 3
2 3 3
1 2 2
--------------
Level [0, 1]=
1 2 2
1 2 2
0 1 1
--------------
Level [1, 1]=
2 3 3
2 3 3
1 2 2
--------------
Level [2, 1]=
3 4 4
3 4 4
2 3 3
--------------
Level [3, 1]=
3 4 4
3 4 4
2 3 3
--------------
Level [0, 2]=
1 2 2
1 2 2
0 1 1
--------------
Level [1, 2]=
2 3 3
2 3 3
1 2 2
--------------
Level [2, 2]=
3 4 4
3 4 4
2 3 3
--------------
Level [3, 2]=
3 4 5
3 4 4
2 3 3
--------------
```

```
Table 5.4.8
Dim=4, delta=5:
Level [0, 0]=
0 1 1
0 0 0
--------------
Level [1, 0]=
1 2 2
0 1 1
--------------
Level [2, 0]=
2 3 3
1 2 2
--------------
Level [3, 0]=
2 3 3
1 2 2
--------------
Level [4, 0]=
2 3 3
1 2 2
--------------
Level [0, 1]=
1 2 2
0 1 1
--------------
Level [1, 1]=
2 3 3
1 2 2
--------------
Level [2, 1]=
3 4 4
2 3 3
--------------
Level [3, 1]=
3 4 4
2 3 3
--------------
Level [4, 1]=
3 4 4
2 3 3
--------------
Level [0, 2]=
1 2 2
0 1 1
--------------
Level [1, 2]=
```

```
2 3 3
1 2 2
--------------
Level [2, 2]=
3 4 4
2 3 3
--------------
Level [3, 2]=
3 4 4
2 3 3
--------------
Level [4, 2]=
3 4 5
2 3 4
--------------
Table 5.4.9
Dim=4, delta=5:
Level [0, 0]=
0 1 1
0 0 0
--------------
Level [1, 0]=
1 2 2
0 1 1
--------------
Level [2, 0]=
2 3 3
1 2 2
--------------
Level [3, 0]=
2 3 3
1 2 2
--------------
Level [0, 1]=
1 2 2
0 1 1
--------------
Level [1, 1]=
2 3 3
1 2 2
--------------
Level [2, 1]=
3 4 4
2 3 3
--------------
Level [3, 1]=
3 4 4
```

```
2 3 3
--------------
Level [0, 2]=
1 2 2
0 1 1
--------------
Level [1, 2]=
2 3 3
1 2 2
--------------
Level [2, 2]=
3 4 4
2 3 3
--------------
Level [3, 2]=
3 4 5
2 3 4
--------------
Table 5.4.10
Dim=4, delta=5:
Level [0, 0]=
0 1 1
0 1 1
0 0 0
--------------
Level [1, 0]=
1 2 2
1 2 2
0 1 1
--------------
Level [2, 0]=
2 3 3
2 3 3
1 2 2
--------------
Level [0, 1]=
1 2 2
1 2 2
0 1 1
--------------
Level [1, 1]=
2 3 3
2 3 3
1 2 2
--------------
Level [2, 1]=
3 4 4
```

```
3 4 4
2 3 3
--------------
Level [0, 2]=
1 2 3
1 2 2
0 1 1
--------------
Level [1, 2]=
2 3 4
2 3 3
1 2 2
--------------
Level [2, 2]=
3 4 5
3 4 4
2 3 3
--------------
Table 5.4.11
Dim=4, delta=5:
Level [0, 0]=
0 1 1
0 0 0
--------------
Level [1, 0]=
1 2 2
0 1 1
--------------
Level [2, 0]=
2 3 3
1 2 2
--------------
Level [0, 1]=
1 2 2
0 1 1
--------------
Level [1, 1]=
2 3 3
1 2 2
--------------
Level [2, 1]=
3 4 4
2 3 3
--------------
Level [0, 2]=
1 2 3
0 1 2
```

```
--------------
Level [1, 2]=
2 3 4
1 2 3
--------------
Level [2, 2]=
3 4 5
2 3 4
--------------
Table 5.4.12
Dim=4, delta=5:
Level [0, 0]=
0 1 1
0 0 0
--------------
Level [1, 0]=
1 2 2
0 1 1
--------------
Level [2, 0]=
2 3 3
1 2 2
--------------
Level [3, 0]=
2 3 3
1 2 2
--------------
Level [4, 0]=
2 3 4
1 2 3
--------------
Level [0, 1]=
1 2 2
0 1 1
--------------
Level [1, 1]=
2 3 3
1 2 2
--------------
Level [2, 1]=
3 4 4
2 3 3
--------------
Level [3, 1]=
3 4 4
2 3 3
--------------
```

```
Level [4, 1]=
3 4 5
2 3 4
--------------
Table 5.4.13
Dim=4, delta=5:
Level [0, 0]=
0 1 1
0 0 0
--------------
Level [1, 0]=
1 2 2
0 1 1
--------------
Level [2, 0]=
2 3 3
1 2 2
--------------
Level [3, 0]=
2 3 4
1 2 3
--------------
Level [0, 1]=
1 2 2
0 1 1
--------------
Level [1, 1]=
2 3 3
1 2 2
--------------
Level [2, 1]=
3 4 4
2 3 3
--------------
Level [3, 1]=
3 4 5
2 3 4
--------------
Table 5.4.14
Dim=4, delta=5:
Level [0, 0]=
0 1
0 1
0 0
--------------
Level [1, 0]=
1 2
```

```
1 2
0 1
--------------
Level [2, 0]=
2 3
1 2
0 1
--------------
Level [0, 1]=
1 2
1 2
0 1
--------------
Level [1, 1]=
2 3
2 3
1 2
--------------
Level [2, 1]=
3 4
2 3
1 2
--------------
Level [0, 2]=
2 3
1 2
0 1
--------------
Level [1, 2]=
3 4
2 3
1 2
--------------
Level [2, 2]=
4 5
3 4
2 3
--------------
Table 5.4.15
Dim=4, delta=5:
Level [0, 0]=
0 1
0 0
--------------
Level [1, 0]=
1 2
0 1
```

```
--------------
Level [2, 0]=
2 3
1 2
--------------
Level [3, 0]=
2 3
1 2
--------------
Level [4, 0]=
3 4
2 3
--------------
Level [0, 1]=
1 2
0 1
--------------
Level [1, 1]=
2 3
1 2
--------------
Level [2, 1]=
3 4
2 3
--------------
Level [3, 1]=
3 4
2 3
--------------
Level [4, 1]=
4 5
3 4
--------------
Table 5.4.16
Dim=4, delta=5:
Level [0, 0]=
0 1
0 0
--------------
Level [1, 0]=
1 2
0 1
--------------
Level [2, 0]=
2 3
1 2
--------------
```

```
Level [3, 0]=
3 4
2 3
--------------
Level [0, 1]=
1 2
0 1
--------------
Level [1, 1]=
2 3
1 2
--------------
Level [2, 1]=
3 4
2 3
--------------
Level [3, 1]=
4 5
3 4
--------------
Table 5.4.17
Dim=4, delta=5:
Level [0, 0]=
0 1
0 0
--------------
Level [1, 0]=
1 2
0 1
--------------
Level [2, 0]=
2 3
1 2
--------------
Level [0, 1]=
1 2
0 1
--------------
Level [1, 1]=
2 3
1 2
--------------
Level [2, 1]=
3 4
2 3
--------------
Level [0, 2]=
```

```
2 3
1 2
--------------
Level [1, 2]=
3 4
2 3
--------------
Level [2, 2]=
4 5
3 4
```

B.5 GapFunctions with $r = 5$

```
                        delta := 5
                      dimension := 5
-------------------------------------------------
Total number of 5-dimensional gap functions: 5
-------------------------------------------------
Table 5.5.1
Dim=5, delta=5:
Level [0, 0, 0]=
0 1 1
0 1 1
0 0 0
--------------
Level [1, 0, 0]=
1 2 2
1 2 2
0 1 1
--------------
Level [2, 0, 0]=
1 2 2
1 2 2
0 1 1
--------------
Level [0, 1, 0]=
1 2 2
1 2 2
0 1 1
--------------
Level [1, 1, 0]=
2 3 3
2 3 3
1 2 2
--------------
```

```
Level [2, 1, 0]=
2 3 3
2 3 3
1 2 2
--------------
Level [0, 2, 0]=
1 2 2
1 2 2
0 1 1
--------------
Level [1, 2, 0]=
2 3 3
2 3 3
1 2 2
--------------
Level [2, 2, 0]=
2 3 3
2 3 3
1 2 2
--------------
Level [0, 0, 1]=
1 2 2
1 2 2
0 1 1
--------------
Level [1, 0, 1]=
2 3 3
2 3 3
1 2 2
--------------
Level [2, 0, 1]=
2 3 3
2 3 3
1 2 2
--------------
Level [0, 1, 1]=
2 3 3
2 3 3
1 2 2
--------------
Level [1, 1, 1]=
3 4 4
3 4 4
2 3 3
--------------
Level [2, 1, 1]=
3 4 4
```

```
3 4 4
2 3 3
--------------
Level [0, 2, 1]=
2 3 3
2 3 3
1 2 2
--------------
Level [1, 2, 1]=
3 4 4
3 4 4
2 3 3
--------------
Level [2, 2, 1]=
3 4 4
3 4 4
2 3 3
--------------
Level [0, 0, 2]=
1 2 2
1 2 2
0 1 1
--------------
Level [1, 0, 2]=
2 3 3
2 3 3
1 2 2
--------------
Level [2, 0, 2]=
2 3 3
2 3 3
1 2 2
--------------
Level [0, 1, 2]=
2 3 3
2 3 3
1 2 2
--------------
Level [1, 1, 2]=
3 4 4
3 4 4
2 3 3
--------------
Level [2, 1, 2]=
3 4 4
3 4 4
2 3 3
```

```
--------------
Level [0, 2, 2]=
2 3 3
2 3 3
1 2 2
--------------
Level [1, 2, 2]=
3 4 4
3 4 4
2 3 3
--------------
Level [2, 2, 2]=
3 4 5
3 4 4
2 3 3
--------------
Table 5.5.2
Dim=5, delta=5:
Level [0, 0, 0]=
0 1 1
0 0 0
--------------
Level [1, 0, 0]=
1 2 2
0 1 1
--------------
Level [2, 0, 0]=
1 2 2
0 1 1
--------------
Level [0, 1, 0]=
1 2 2
0 1 1
--------------
Level [1, 1, 0]=
2 3 3
1 2 2
--------------
Level [2, 1, 0]=
2 3 3
1 2 2
--------------
Level [0, 2, 0]=
1 2 2
0 1 1
--------------
Level [1, 2, 0]=
```

```
2 3 3
1 2 2
--------------
Level [2, 2, 0]=
2 3 3
1 2 2
--------------
Level [0, 0, 1]=
1 2 2
0 1 1
--------------
Level [1, 0, 1]=
2 3 3
1 2 2
--------------
Level [2, 0, 1]=
2 3 3
1 2 2
--------------
Level [0, 1, 1]=
2 3 3
1 2 2
--------------
Level [1, 1, 1]=
3 4 4
2 3 3
--------------
Level [2, 1, 1]=
3 4 4
2 3 3
--------------
Level [0, 2, 1]=
2 3 3
1 2 2
--------------
Level [1, 2, 1]=
3 4 4
2 3 3
--------------
Level [2, 2, 1]=
3 4 4
2 3 3
--------------
Level [0, 0, 2]=
1 2 2
0 1 1
--------------
```

```
Level [1, 0, 2]=
2 3 3
1 2 2
--------------
Level [2, 0, 2]=
2 3 3
1 2 2
--------------
Level [0, 1, 2]=
2 3 3
1 2 2
--------------
Level [1, 1, 2]=
3 4 4
2 3 3
--------------
Level [2, 1, 2]=
3 4 4
2 3 3
--------------
Level [0, 2, 2]=
2 3 3
1 2 2
--------------
Level [1, 2, 2]=
3 4 4
2 3 3
--------------
Level [2, 2, 2]=
3 4 5
2 3 4
--------------
Table 5.5.3
Dim=5, delta=5:
Level [0, 0, 0]=
0 1 1
0 0 0
--------------
Level [1, 0, 0]=
1 2 2
0 1 1
--------------
Level [0, 1, 0]=
1 2 2
0 1 1
--------------
Level [1, 1, 0]=
```

```
2 3 3
1 2 2
--------------
Level [0, 2, 0]=
1 2 2
0 1 1
--------------
Level [1, 2, 0]=
2 3 3
1 2 2
--------------
Level [0, 0, 1]=
1 2 2
0 1 1
--------------
Level [1, 0, 1]=
2 3 3
1 2 2
--------------
Level [0, 1, 1]=
2 3 3
1 2 2
--------------
Level [1, 1, 1]=
3 4 4
2 3 3
--------------
Level [0, 2, 1]=
2 3 3
1 2 2
--------------
Level [1, 2, 1]=
3 4 4
2 3 3
--------------
Level [0, 0, 2]=
1 2 2
0 1 1
--------------
Level [1, 0, 2]=
2 3 3
1 2 2
--------------
Level [0, 1, 2]=
2 3 3
1 2 2
--------------
```

```
Level [1, 1, 2]=
3 4 4
2 3 3
--------------
Level [0, 2, 2]=
2 3 4
1 2 3
--------------
Level [1, 2, 2]=
3 4 5
2 3 4
--------------
Table 5.5.4
Dim=5, delta=5:
Level [0, 0, 0]=
0 1 1
0 0 0
--------------
Level [1, 0, 0]=
1 2 2
0 1 1
--------------
Level [0, 1, 0]=
1 2 2
0 1 1
--------------
Level [1, 1, 0]=
2 3 3
1 2 2
--------------
Level [0, 0, 1]=
1 2 2
0 1 1
--------------
Level [1, 0, 1]=
2 3 3
1 2 2
--------------
Level [0, 1, 1]=
2 3 3
1 2 2
--------------
Level [1, 1, 1]=
3 4 4
2 3 3
--------------
Level [0, 0, 2]=
```

```
1 2 3
0 1 2
--------------
Level [1, 0, 2]=
2 3 4
1 2 3
--------------
Level [0, 1, 2]=
2 3 4
1 2 3
--------------
Level [1, 1, 2]=
3 4 5
2 3 4
--------------
Table 5.5.5
Dim=5, delta=5:
Level [0, 0, 0]=
0 1
0 0
--------------
Level [1, 0, 0]=
1 2
0 1
--------------
Level [0, 1, 0]=
1 2
0 1
--------------
Level [1, 1, 0]=
2 3
1 2
--------------
Level [0, 2, 0]=
2 3
1 2
--------------
Level [1, 2, 0]=
3 4
2 3
--------------
Level [0, 0, 1]=
1 2
0 1
--------------
Level [1, 0, 1]=
2 3
```

```
1 2
--------------
Level [0, 1, 1]=
2 3
1 2
--------------
Level [1, 1, 1]=
3 4
2 3
--------------
Level [0, 2, 1]=
3 4
2 3
--------------
Level [1, 2, 1]=
4 5
3 4
```

B.6 GapFunctions with $r = 6$

```
                        delta := 5
                      dimension := 6
-------------------------------------------------
Total number of 6-dimensional gap functions: 1
-------------------------------------------------
Table 5.6.1
Dim=6, delta=5:
Level [0, 0, 0, 0]=
0 1
0 0
--------------
Level [1, 0, 0, 0]=
1 2
0 1
--------------
Level [0, 1, 0, 0]=
1 2
0 1
--------------
Level [1, 1, 0, 0]=
2 3
1 2
--------------
Level [0, 0, 1, 0]=
1 2
```

```
0 1
--------------
Level [1, 0, 1, 0]=
2 3
1 2
--------------
Level [0, 1, 1, 0]=
2 3
1 2
--------------
Level [1, 1, 1, 0]=
3 4
2 3
--------------
Level [0, 0, 0, 1]=
1 2
0 1
--------------
Level [1, 0, 0, 1]=
2 3
1 2
--------------
Level [0, 1, 0, 1]=
2 3
1 2
--------------
Level [1, 1, 0, 1]=
3 4
2 3
--------------
Level [0, 0, 1, 1]=
2 3
1 2
--------------
Level [1, 0, 1, 1]=
3 4
2 3
--------------
Level [0, 1, 1, 1]=
3 4
2 3
--------------
Level [1, 1, 1, 1]=
4 5
3 4
```