Associated Graded Rings for Numerical Monoid Rings

A Thesis submitted to the faculty of

San Francisco State University

In partial satisfaction of the

requirements for

the Degree

Masters of Arts

in

Mathematics

by

Curtis Marshall Olinger

San Francisco, California

May 2022

Abstract

Numerical monoids, i.e., the additive submonoids of non-negative integers with finite complement, are central objects of study at the crossroads of combinatorics, additive number theory, and commutative algebra. A well-known invariant, attached to a numerical monoid, is the Frobenius number. It is the largest integer not in the monoid. Despite many partial results, no general formula is known for the Frobenius number in terms of given generators of the monoid. We introduce a new invariant of a numerical monoid, called the second Frobenius number. It carries important structural information on the monoid. A commutative algebra perspective explains the relationship between the original and the second Frobenius numbers: both are the threshold points for stabilization of Hilbert functions – for the monoid ring itself in the classical setting and for the associated graded algebra in our setting. Our main theoretical result is an explicit upper bound for the second Frobenius number in terms of given generators of the monoid. We also develop several algorithms for computing the second Frobenius number and present many computational results, strongly suggesting that the Hilbert functions may be stabilizing much faster than our theoretical bound.

Acknowledgments

I would like to thank my wife for encouraging me to undertake a master's degree and supporting me through the process. I honestly could not have done it without her. I would like to thank my advisor Dr. Joseph Gubeladze who kindly and humbly worked with me on this thesis. He always took the time to teach me and to answer the many questions I had. The generosity of his time is deeply appreciated. I would also like to thank David Erickson for his help solving some of the algorithmic challenges we faced.

Table of Contents

Introduction

Ferdinand Georg Frobenius was a German mathematician born February 14, 1877 in a suburb of Berlin. He worked on diverse fields such as elliptic functions, differential equations, number theory, and group theory. He has numerous mathematical ideas named after him but one in particular will be the focus of this paper.

In the early part of the 20th century, Frobenius proposed the *Diophantine Frobenius Problem* which would motivate the study of numerical monoids. The problem asks what is the largest positive integer (called the *Frobenius number*) that is *not* representable as a nonnegative integer linear combination of relatively prime positive integers? For example, if we take the subset of the natural numbers $\{5, 7, 9\}$ and consider all possible combinations of these numbers if we can only add them (including repetitions), then we get the following set

$$\{5, 7, 9, 10, 12, 14, \rightarrow\}$$

where the symbol $\rightarrow$ means that every integer greater than 14 is in this set. We can see the largest natural number not in this set is 13.

What Frobenius ended up doing by proposing this question was to motivate the study of the gaps in the natural numbers. This study is a fascinating tour de force of linear algebra, number theory, and abstract algebra; see [RG09; Ram05; CGO20; ADG20]. The Frobenius number problem has a ring theoretical interpretation which we will explain after introducing several algebraic concepts.

In this paper we denote the nonnegative integers by $\mathbb{Z}_+$ and $\mathbb{N}$ to denote the natural numbers $\{1, 2, 3, \ldots\}$.

A *numerical monoid* M is a subset of the nonnegative integers that contains the additive identity 0, is closed under addition and has a finite complement (possibly empty) in the nonnegative integers. Every numerical monoid M can be written (non uniquely in general) as

$$\mathbb{Z}_+ m_1 + \cdots + \mathbb{Z}_+ m_e := \{a_1 m_1 + \cdots + a_e m_e \mid a_1, \ldots, a_e \in \mathbb{Z}_+\}$$

for some relatively prime natural numbers $m_1, \ldots, m_e$, called *generators* of M.

For a field $\mathbf{k}$ and a numerical monoid M, the *monoid algebra* $\mathbf{k}[M]$ can be thought of as the $\mathbf{k}$-subalgebra of the univariate polynomial ring $\mathbf{k}[X]$, consisting of the polynomials whose support monomials are of the form λX^m, where $\lambda \in \mathbf{k}$ and $m \in M$. In other words, $\mathbf{k}[M]$ is the $\mathbf{k}$-linear span of the monomials X^m, $m \in M$: this $\mathbf{k}$-vector space contains $\mathbf{k}$ and is closed under the usual polynomial multiplication. There is also a surjective $\mathbf{k}$-algebra homomorphism from the multivariate polynomial ring $\mathbf{k}[X_1, \ldots, X_e]$ to $\mathbf{k}[M]$, namely

$$\mathbf{k}[X_1, \ldots, X_e] \to \mathbf{k}[M],$$

$$X_i \mapsto X^{m_i},$$

$$i = 1, \ldots, e.$$

In the case $\mathbf{k}$ is algebraically closed, this surjection via the *Hilbert Nullstellensatz* induces an embedding of the *monomial variety*, corresponding to $\mathbf{k}[M]$, in the *affine space* of dimension e. Correspondingly, e is called the *embedding dimension*, which also explains our notation.

The algebra $\mathbf{k}[M]$ is graded with respect to the degrees in X:

$$\mathbf{k}[M] = \mathbf{k} \oplus A_1 \oplus A_2 \oplus \cdots$$

and we have the resulting Hilbert function $H_{\mathbf{k}[M]}(i) = \dim_{\mathbf{k}} A_i$. For every i, we have $H_{\mathbf{k}[M]}(i) = 0$ or 1 and, since $\mathbb{Z}_+ \setminus M$ is finite, the function $H_{\mathbf{k}[M]}(-)$ eventually becomes constant with value 1. The Frobenius number of $m_1, \ldots, m_e$ is then the largest value of i, for which $H_{\mathbf{k}[M]}(i) = 0$. In other words, the Frobenius number problem asks to determine when the mentioned Hilbert function stabilizes.

Our work is about the stabilization of the Hilbert function of another graded algebra, also naturally associated to M – the *associated graded algebra* $\mathrm{gr}(\mathbf{k}[M])$ of $\mathbf{k}[M]$ with respect to the maximal monomial ideal in $\mathbf{k}[M]$. It determines a *flat deformation* of $\mathbf{k}[M]$ and the Hilbert function of $\mathrm{gr}(\mathbf{k}[M])$ carries vital information on the additive structure of M. In plain monoid terms, $H_{\mathrm{gr}(\mathbf{k}[M])}(i)$ is the number of elements of M that can be written as sums of i generators and can't be written as sums of more than i generators.

It follows from dimension theory in commutative algebra that $H_{\mathrm{gr}(\mathbf{k}[M])}(-)$ eventually stabilizes, just like $H_{\mathbf{k}[M]}(-)$, and the determination where this happens is a nontrivial challenge. We call the stabilization index in this Hilbert function the *second Frobenius number* of M, denoted by $\mathbf{F}'(M)$. To the best of our knowledge, these numbers have not been studied, although the asymptotic behavior of $H_{\mathrm{gr}(\mathbf{k}[M])}$ is known [ONe17]. The commutative algebra approach only implies the existence of $\mathbf{F}'(M)$, but does not offer an upper bound.

Our main theoretical result is an explicit upper bound of the second Frobenius number in terms of the generators of M (Theorem 4.7). In the second part of the work we implement several algorithms for computing $\mathbf{F}'(M)$ and use it to develop computational data on many examples of numerical monoids. These computations provide strong evidence that $\mathbf{F}'(M)$ is considerably smaller than the theoretical upper bound in Theorem 4.7.

We expect that the second Frobenius number is at least as interesting as the original Frobenius number and it provides a fertile ground for exploration of numerical monoids from this novel perspective.

Chapter 1

Semigroups and monoids

1.1 Semigroups

Our study of numerical monoids starts with that of semigroups. A *semigroup* is a pair $(S, +)$ with S a set and $+$ an associative binary operation on S. Semigroups do not need inverses nor an identity element like a group does. We will assume all semigroups in this paper are commutative and we will omit the binary operation $+$ and denote the semigroup just by S. Thus $\mathbb{N}$ is a semigroup under addition as is the set of positive even integers $2\mathbb{N}$.

A *subsemigroup* is a subset of a semigroup S that is closed under the operation of S. Thus, $2\mathbb{N}$ is a subsemigroup of $\mathbb{N}$.

The intersection of any number of subsemigroups is a subsemigroup. To see this, one must show that the intersection of a set of subsemigroups of S is closed under the operation of S. Each element in the intersection is in every subsemigroup and each subsemigroup is closed under the operation of S by definition. Thus the intersection is closed which makes the intersection a subsemigroup.

Let S be a semigroup and let $s_1, \ldots, s_k$ be any collection of elements of S. *The smallest subsemigroup that contains* $s_1, \ldots, s_k$ is the intersection of all subsemigroups that contain $s_1, \ldots, s_k$. We denote this set $\langle s_1, \ldots, s_k \rangle$ and we have

$$\langle s_1, \ldots, s_k \rangle = \left\{ \sum_{i=1}^{k} a_i s_i \mid a_i \in \mathbb{Z}_+ \text{ and at least one } a_i \neq 0 \right\}.$$

Remember it is not necessary for the additive identity to be an element of a semigroup.

If $\langle s_1, \ldots, s_k \rangle = S$ for some $k \in \mathbb{N}$ then we say that S is generated by $s_1, \ldots, s_k$ and S is *finitely generated*. When $s \in \langle s_1, \ldots, s_k \rangle$ is written as $s = \sum_{i=1}^{k} a_i s_i$ with $a_i \in \mathbb{N}$, then we call this sum a *representation* of $s \in S$.

1.2 Monoids

A *monoid* M is a semigroup with a neutral element 0. A subset N of a monoid M is a *submonoid* of M if it is a subsemigroup that contains the neutral element. Note that the set $\{0\}$ is a submonoid of M and is called the *trivial submonoid*.

As with the case with semigroups, the intersection of any number of submonoids is a submonoid. Thus, for any subset N of a monoid M, *the smallest submonoid containing* N is the intersection of all submonoids of M that contain N. This is also denoted by $\langle N \rangle$. Whether this is a subsemigroup or a submonoid will be clear from context (just check if it contains 0). Since $\langle N\rangle$ is closed under the operation of M, all elements have the form

$$\langle N\rangle = \left\{\sum_{i=1}^{k} a_i s_i \mid a_i \in \mathbb{Z}_+,\ s_i \in N\right\}.$$

Notice that

$$\langle\emptyset\rangle = \{0\} = \langle 0\rangle$$

since $\{0\}$ and the empty set $\emptyset$ are subsets of every submonoid. We will now discuss an important submonoid called a numerical monoid.

Chapter 2

Numerical monoids and the Frobenius number

2.1 Numerical monoids

In this paper we are only interested in the submonoids of the nonnegative integers. These submonoids give a rich source of material to study. We therefore come to an important definition.

Definition 2.1. A *numerical monoid* is a submonoid of $\mathbb{Z}_+$ with finite complement in $\mathbb{Z}_+$.

The set $\{0, 6, 7, 8, 9, \rightarrow\}$ is a numerical monoid, whereas the set of even nonnegative integers $2\mathbb{Z}_+$ is not because its complement in $\mathbb{Z}_+$ is the infinite set of odd integers.

Since the complement of a numerical monoid M in $\mathbb{Z}_+$ is finite, there must be a $m_0 \in M$, such that for all $m > m_0$, $m+1 \in M$. In other words, the sequence of numerical monoid elements includes all consecutive integers after some point. In this light, the following is a proposition that can be used as the definition of a numerical monoid.

Proposition 2.2. *Let M be a submonoid of $\mathbb{Z}_+$. Let G be the subgroup of $\mathbb{Z}$ generated by M. Then M is a numerical monoid if and only if $1 \in G$, i.e., $G = \mathbb{Z}$.*

Proof. Let M be a numerical monoid. Since the complement of M in $\mathbb{Z}_+$ has finite cardinality, there exists $m \in M$ such that $m+1 \in M$. Thus for $m, m+1 \in G$, we have $(m+1) - m = 1 \in G$.

Now assume $1 \in G$ which implies there exist $m \in M$ such that $(m+1) - m = 1$. Thus $m+1 \in M$. Now we have to show M has a finite complement in $\mathbb{Z}_+$. We claim that if $n \geq (m-1)(m+1)$, then $n \in M$.

Let $n \geq (m-1)(m+1)$ and by the division algorithm there exists unique $q, r \in \mathbb{Z}$ such that $n = qm + r$, where $0 \leq r < m$. Notice that $(m-1)(m+1) = (m-1)m + (m-1)$. So $n = qm + r \geq (m-1)m + (m-1)$. Then $q \geq m-1$ and $m > r$ implies that $m - 1 \geq r$. Thus $q > r$. If we write, $n = qm + r = (q-r)m + r(m+1)$, then $q - r > 0$ and $r > 0$. Therefore we have written n as a nonnegative integer combination of m and $m+1$ which means that $n \in M$. Furthermore, any $n \geq (m-1)(m+1) \in M$. Thus the complement of M in $\mathbb{Z}_+$ must be finite. $\square$

Now we will focus on numerical monoids, given by generators.

Proposition 2.3. *Let A be a nonempty subset of $\mathbb{Z}_+$. Then $\langle A\rangle$ is a numerical monoid if and only if* $\gcd(A) = 1$.

Proof. The subgroup of $\mathbb{Z}$, generated by $\langle A\rangle$ is the same as the subgroup generated by A. According to Proposition 2.2, the latter is $\mathbb{Z}$ if and only if $\langle A\rangle$ is a numerical monoid which, in turn, is equivalent to $\gcd(A) = 1$. $\square$

Proposition 2.4. *Let M be a nontrivial submonoid of $\mathbb{Z}_+$. Then M is isomorphic to a numerical monoid.*

Proof. Let $d = \gcd(M)$ and let $S = \{\frac{m}{d} : m \in M\}$. Thus the $\gcd(S) = 1$ and by Proposition 2.3, S is a numerical monoid and the map $\phi : M \to S$ by $\phi(m) = \frac{m}{d}$ is an isomorphism. $\square$

For example $2\mathbb{Z}_+$ is a submonoid of $\mathbb{Z}_+$. All elements have the form $2x$ for some $x \in \mathbb{Z}_+$. Thus for $m \in 2\mathbb{Z}_+$, $\phi(m) = \phi(2x) = \frac{2x}{x} = x$. Therefore $2\mathbb{Z}_+ \cong \mathbb{Z}_+$.

Now some terminology about what it means to add two sets together. Let $A, B \subset \mathbb{Z}_+$. Then

$$A + B = \{a + b : a \in A \text{ and } b \in B\}.$$

Let M be a numerical monoid and define $M^* = M \setminus \{0\}$. Then the set $M^* + M^*$ is the set of elements in M that are the sum of two nonzero elements in M. For example if $M = \langle 2, 3\rangle$, then

$$M^* + M^* = \{4, 5, 6, 7, \ldots\}.$$

Notice that the generators of M are absent from this set.

Lemma 2.5. *Let M be a submonoid of $\mathbb{Z}_+$. Then $M^* \setminus (M^* + M^*)$ is a system of generators of M. Moreover, every system of generators of M contains $M^* \setminus (M^* + M^*)$.*

Proof. Let $m \in M^*$. We want to show that there exist $m_1, \ldots, m_k \in M^* \setminus (M^* + M^*)$ such that $m = a_1 m_1 + \cdots + a_k m_k$ for some $a_i \in \mathbb{Z}_+$. If $m \notin M^* \setminus (M^* + M^*)$ then there exist $x, y \in M^*$ such that $m = x + y$. Repeat this procedure with x and y and so on. The process must terminate because of the Well Ordering Principle: after each step we get smaller summands. This shows that $M^* \setminus (M^* + M^*) \neq \emptyset$ and, moreover, $M^* \setminus (M^* + M^*)$ generates M.

That every generating set of M must contain $M^* \setminus (M^* + M^*)$ is straightforward [BG09, Chapter 2.A]. □

Definition 2.6. For a numerical monoid M, its *Hilbert basis* is the set of *indecomposable elements*, i.e., $\mathrm{Hilb}(M) = M^* \setminus (M^* + M^*)$

2.2 Decomposition length

Let M be a numerical monoid generated by a set $\{m_1, \ldots, m_e\}$ and $m_1 < \cdots < m_e$. For any element $m \in M$ and a representation $m = a_1 m_1 + \cdots + a_e m_e$, the e-tuple $(a_1, \ldots, a_e)$ is a *decomposition of* $m \in M$.

Next we introduce the *maximal decomposition lengths*. For any $m \in M \setminus \{0\}$, a representation $m = \sum_{i=1}^{e} a_i m_i$, with $a \in \mathbb{Z}_+$, is called a *maximal decomposition* if for any other representation $m = \sum_{i=1}^{e} b_i m_i$, with $b_i \in \mathbb{Z}_+$, one has

$$\sum_{i=1}^{e} b_i \leq \sum_{i=1}^{e} a_i.$$

Correspondingly, the *maximal decomposition length*, or simply the *length*, of an element $m \in M \setminus \{0\}$ is defined by

$$\mathbf{l}(m) = \sum_{i=1}^{e} a_i,$$

where $m = \sum_{i=1}^{e} a_i m_i$ is any maximal decomposition of $m \in M$. A *maximal decomposition of* $m \in M$ is the e-tuple $(a_1, \ldots, a_e)$, whereas $\sum_{i=1}^{e} a_i m_i$ is a maximal decomposition.

For the number of elements in M that have a particular length $k \in \mathbb{N}$ we write

$$d_k(M) = \#\{m \in M \mid \mathbf{l}(m) = k\}.$$

For the set of all maximal decompositions for a length $k \in \mathbb{N}$, we write

$$\mathbf{dec}_k(M) = \{(a_1, \ldots, a_e) \mid m = \sum_{i=1}^{e} a_i m_i \text{ and } \mathbf{l}(m) = k\}.$$

Finally, we put

$$\mathbf{dec}(M) = \bigcup_{k=1}^{\infty} \mathbf{dec}_k(M).$$

Next proposition explain that the numbers $d_k(M)$ and $\mathbf{dec}_k(M)$ are independent of the choices of the generating set $\{m_1, \ldots, m_e\}$.

Proposition 2.7. *For a numerical monoid M and a natural number k, the numbers $d_k(M)$ and $\#\boldsymbol{dec}_k(M)$ with respect to a generating set $\{m_1, \ldots, m_e\}$ are the same as with respect to* Hilb(M).

Proof. For every element $m \in M$, in any representation of m in terms of the generators $m_1, \dots, m_e$ we can further decompose the summands into elements of $\mathrm{Hilb}(M)$. On the other hand, by Lemma 2.5, $\mathrm{Hilb}(M) \subset \{m_1, \dots, m_e\}$. This shows that the number $d_k(m)$ is computed in terms of representations of m via $\mathrm{Hilb}(M)$ and, also, in any maximal length representation in terms of $m_1, \dots, m_e$ only indecomposable elements are used. $\square$

The numerical sequences $\{d_k(M)\}_{k=0}^{\infty}$, and $\{\#\mathbf{dec}_k(M)\}_{k=0}^{\infty}$ are our primary objects of study.

The following lemma is a special case – the rank one case – of the *Gordan Lemma*, concerning *affine submonoids* of arbitrary rank [BG09, Chapter 2].

Lemma 2.8. *Let $M \subset \mathbb{Z}_+$ be a numerical monoid and $m = \min(M \backslash \{0\})$. Then $\# \mathrm{Hilb}(M) < m$. In particular, M is finitely generated.*

Proof. Since $\#(\mathbb{Z}_+ \backslash M) < \infty$, every residue class $\mod m$ occurs in M. Let $m = m_0, m_1, \dots, m_{m-1}$ be the smallest elements of M, satisfying $m_i = i \mod m$. Then every element $n \in M$ can be (uniquely) written as

$$n = qm + m_i, \quad q \in \mathbb{Z}_+, \quad n = i \mod m.$$

In particular, $M = \langle m_0, m_1, \dots, m_{m-1} \rangle$. $\square$

2.3 Frobenius number

The Frobenius number of a numerical monoid M is an active research area [Ram05]. The main problem is to express $\mathbf{F}(M)$, or at least an optimal upper bound in terms of a given generating set of M. Below we give a short synopses of some of the highlights in the field. For $m_1, \ldots, m_e \in \mathbb{N}$ with $\gcd(m_1, \ldots, m_e) = 1$ and $m_1 < \cdots < m_e$, we denote

$$\mathbf{F}(m_1, \ldots, m_e) = \mathbf{F}(\langle m_1, \ldots, m_e \rangle).$$

(a) $\mathbf{F}(m_1, m_2) = m_1 m_2 - m_1 - m_2$ and there are $\frac{1}{2}(m_1 - 1)(m_2 - 1)$ non-representable positive integers;

(b) $\mathbf{F}(m_1, \ldots, m_e) < m_1 m_e$;

(c) The lower bound for the Frobenius number with embedding dimension 3 is given by

$$F(m_1, m_2, m_3) + m_1 + m_2 + m_3 \geq \sqrt{m_1 m_2 m_3};$$

(d) For the following arithmetic sequence we have

$$F(m_1, m_1 + m_2, m_1 + 2m_2, \ldots, m_1 + m_2 m_3) = \left(\left\lfloor \frac{m_1 - 2}{m_3} \right\rfloor \right) m_1 + m_2(m_1 - 1);$$

(e) For the following geometric sequence we have

$$F(m_1^k, m_1^{k-1} m_2, m_1^{k-2} m_2^2, \ldots, m_2^k) =$$
$$m_2^{k-1}(m_1 m_2 - m_1 - m_2) + \frac{m_1^2 (n-1)(m_1^{k-1} - m_2^{m_2 - 1})}{m_1 - m_2};$$

(f) *Arnold's conjecture.* Let $T = \sum_{i=1}^{e} |m_i|$. Thus T is the ℓ^1 norm of the generators. One of the most famous open problems in the field, Arnold's conjecture, says that $\mathbf{F}(M)$ increases like $T^{1+1/(e-1)}$ [Isk11, p. 526]. Arnold also conjectured that for "average behavior", $\mathbf{F}(M)$ is

$$F(M) \sim (e-1)!^{\frac{1}{e-1}}(m_1 m_2 \cdots m_e)^{\frac{1}{e-1}} \quad \text{(see [Isk11, p. 526])}.$$

Chapter 3

Commutative algebra of numerical monoid rings

All our rings are assumed to be commutative and unital. The symbol $\mathbf{k}$ will always denote a field and $\mathbf{k}^* = \mathbf{k} \setminus \{0\}$. Also, all our monoids are commutative and ring homomorphisms are assumed to respect the units.

3.1 Algebras

Let $\mathbf{k}$ be a field and A a ring. Then, A is called a $\mathbf{k}$-*algebra* if A contains a isomorphic copy of $\mathbf{k}$ as a subring. For simplicity of notation, we will identify $\mathbf{k}$ with its isomorphic copy in A. Every $\mathbf{k}$-algebra is also a $\mathbf{k}$-vector space. For two $\mathbf{k}$ algebras A and B, a ring homomorphism $f : A \to B$ is a $\mathbf{k}$-*algebra homomorphism* if it is also a $\mathbf{k}$-linear map.

The basic example of a $\mathbf{k}$-algebra is the multivariate polynomial ring $\mathbf{k}[X_1, \ldots, X_n]$. The quotient of a $\mathbf{k}$-algebra by an ideal is also a $\mathbf{k}$-algebra.

A $\mathbf{k}$-algebra A is called *finitely generated* if there is a finite family of elements $\{a_1, \ldots, a_n\} \subset A$, called *generators of* A, such that A is the smallest sub-algebra of A, containing the a_i's. In this case we will write $A = \mathbf{k}[a_1, \ldots, a_n]$. The assignment $X_i \mapsto a_i$, $i = 1, \ldots n$, gives rise to a surjective $\mathbf{k}$-algebra homomorphism

$$\mathbf{k}[X_1, \ldots, X_n] \to A.$$

Hence, the *Isomorphism Theorem* for rings implies that *every* finitely generated $\mathbf{k}$-algebra is isomorphic to a quotient of the form $\mathbf{k}[X_1, \ldots, X_n]/I$ for some natural number $n \in \mathbb{N}$ and an ideal $I \subset \mathbf{k}[X_1, \ldots, X_n]$.

Notice, a (finite) generating set of a $\mathbf{k}$-algebra $\mathbf{k}[a_1, \ldots, a_n]$ is highly non-unique, not even if the a_i are variables and we only consider generating sets of the smallest size. In fact, we have

$$\mathbf{k}[X_1, \ldots, X_n] = \mathbf{k}[\mu_1 X_1 + \lambda_1, \ldots, \mu_n X_n + \lambda_n] =$$
$$\mathbf{k}[X_1, X_2 + F_2(X_1), X_3 + F_3(X_1, X_2), \ldots, X_n + F_n(X_1, \ldots, X_{n-1})],$$

for arbitrary elements

(a) $\mu_i \in \mathbf{k}^*$ and $\nu_i \in \mathbf{k}$, where $i = 1, \ldots, n$,

(b) $F_i(X_1, \ldots, X_{i-1}) \in \mathbf{k}[X_1, \ldots, X_{i-1}]$, where $i = 2, \ldots, n$

3.2 Monoid rings

Let M be a monoid. The monoid algebra $\mathbf{k}[M]$ is the $\mathbf{k}$-vector space over the basis M, where the multiplicative structure is defined by

$$\left(\sum_i \lambda_i m_i\right) \cdot \left(\sum_j \mu_j n_j\right) = \sum_{i,j} (\lambda_i \mu_j)(m_i n_j),$$

where $\lambda_i, \mu_j \in \mathbf{k}$, $m_i, n_j \in M$, and the monoid operation is written multiplicatively. The reader is referred to [BG09, Chapter 2] for generalities on monoid rings.

The monoid ring $\mathbf{k}[M]$ is a $\mathbf{k}$-algebra, where $\mathbf{k}$ embeds to $\mathbf{k}[M]$ via $\lambda \mapsto \lambda \cdot 1$ for the neutral element $1 \in M$ (changed from the additive notation $0 \in M$).

The algebra $\mathbf{k}[M]$ also contains an isomorphic copy of M via the embedding $m \mapsto 1 \cdot m$, where $1 \in \mathbf{k}$.

Notice, $1 \in \mathbf{k}$ and $1 \in M$ are the same unit element of $\mathbf{k}[M]$.

We will write elements of $\mathbf{k}[M]$ as linear combinations $\sum_i \lambda_i m_i$, with the understanding that $1 \in M$ gets identified with $1 \in \mathbf{k}$ (and the monoid operation is written multiplicatively).

The defining universal property of the monoid ring $\mathbf{k}[M]$ is that, for any $\mathbf{k}$-algebra A and any monoid homomorphism $f : M \to A$ with respect to the multiplicative structure of A, there exists a unique $\mathbf{k}$-algebra homomorphism $\mathbf{k}[M] \to A$, extending f. Moreover, this universal property defines the algebra $\mathbf{k}[M]$ uniquely, up to isomorphism.

Example 3.1. The basic examples of a monoid ring is the polynomial rings $\mathbf{k}[(\mathbb{Z}_+)^n] = \mathbf{k}[X_1, \ldots, X_n]$, where the identification of the two $\mathbf{k}$-algebras is through the $\mathbf{k}$-algebra isomorphism, defined by

$$(a_1, \ldots, a_n) \mapsto X_1^{a_1} \cdots X_n^{a_n}.$$

Example 3.2. For a numerical monoid monoid M, the monoid algebra $\mathbf{k}[M]$ can be though of as the subalgebra of the univariate polynomial ring $\mathbf{k}[X]$, consisting of the polynomials, whose reduced forms only involve monomials of the form λX^m, where $\lambda \in \mathbf{k}$ and $m \in M$.

3.3 Embedding dimension and multiplicity

Assume $M \subset \mathbb{Z}_+$ is a numerical monoid, generated by coprime numbers $m_1, \ldots, m_e \in \mathbb{N}$. Then we have the $\mathbf{k}$-algebra homomorphisms:

$$f : \mathbf{k}[X_1, \ldots, X_e] \to \mathbf{k}[M], \quad X_i \mapsto m_i, \quad i = 1, \ldots, e,$$

$$g : \mathbf{k}[X] \to \mathbf{k}[M], \quad X \mapsto m_1.$$

For the corresponding affine varieties (assuming $\mathbf{k}$ is algebraically closed), via *Hilbert Nullstellensatz* [AM69, Chapter 7], f induces an algebraic embedding of the monomial curve $\mathrm{Spec}(\mathbf{k}[M])$ into the affine space $\mathbb{A}^n_{\mathbf{k}}$. This monomial curve is given by

$$\{(t^{m_1}, t^{m_2}, \ldots, t^{m_e}) \mid t \in \mathbf{k}\}$$

and an algebraic surjection from the $\mathrm{Spec}(\mathbf{k}[M])$ to the affine line $\mathbb{A}^1_{\mathbf{k}}$, given by

$$(t^{m_1}, t^{m_2}, \ldots, t^{m_e}) \longmapsto t^{m_1}, \quad t \in \mathbf{k},$$

which is generically of the form 'n points to one point'. Correspondingly, e is called the *embedding dimension* of M with respect to the given generators and m_1 is called the *multiplicity* of M. Notice, according to Lemma 2.5, the multiplicity is independent of the choice of generators. This terminology also explains our use of e for the number of generators of M.

3.4 Graded rings

A *graded* $\mathbf{k}$-algebra A is an algebra, admitting a *direct sum* representation

$$A = A_0 \oplus A_1 \oplus A_2 \oplus \cdots$$

where $A_i \subset A$ is a $\mathbf{k}$-vector subspace, respecting the multiplicative structure as follows: $A_i \cdot A_i \subset A_{i+j}$ for every $i, j \in \mathbb{Z}_+$.

We call A_n the *degree n-homogeneous component* of A. For an element $a \in A_n \setminus \{0\}$, we write $\deg(a) = n$.

A $\mathbf{k}$-algebra can carry several different graded structures. For instance, we can make the polynomial ring $\mathbf{k}[X_1, \ldots, X_n]$ into a graded algebra in infinitely many different ways.

Example 3.3. Every family of natural numbers $c_1, \ldots c_n \in \mathbb{N}$ defines a graded structure

$$\mathbf{k}[X_1, \ldots, X_n] = \mathbf{k} \oplus A_1 \oplus A_2 \oplus \cdots ,$$

via putting $\deg(X_i) = c_i$. In more detail, under this grading we have

$$A_i = \left\{ \sum_t \lambda_t X_1^{a_{t1}} \cdots X_n^{a_{tn}} \;\middle|\; \lambda_t \in \mathbf{k},\ c_1 a_{t1} + \cdots + c_n a_{tn} = i \right\}.$$

When $c_1 = c_2 = \cdots = c_n = 1$, we get the *standard* grading of the polynomial ring.

Let $A = A_1 \oplus A_2 \oplus \cdots$ be a graded $\mathbf{k}$-algebra, such that the $\mathbf{k}$-vector space dimension $\dim_{\mathbf{k}}(A_n)$ is finite for every n. Then the function $H_A : \mathbb{Z}_+ \to \mathbb{Z}_+$, defined by $H_A(n) = \dim_{\mathbf{k}}(A_n)$ is the *Hilbert function* of A (for this grading).

Example 3.4. For the standard grading of $A = \mathbf{k}[X_1, \ldots, X_n]$, we have

$$H_A(i) = \binom{n+i-1}{i} \qquad \text{(see [Eis95, p. 45])}.$$

We will need the following consequence of the general dimension theory of graded rings [AM69, Chapter 11], where $\sqrt{0}$ denotes the ideal of all nilpotent elements – the *nil-radical:*

Lemma 3.5. *Let $A = \mathbf{k} \oplus A_1 \oplus A_2 \oplus \cdots$ be a finitely generated graded $\mathbf{k}$-algebra such that $A/\sqrt{0} \cong \mathbf{k}[X]$ as $\mathbf{k}$-algebras, then A is a one-dimensional ring and, consequently, H_A is eventually a constant function, i.e., $H_A(i) = H_A(i+1)$ for $i > 0$.*

Remark. We point out that dimension theory in commutative algebra does *not* say how large i needs to be to guarantee the equality $H_A(i) = H_A(i+1)$.

For $\mathbf{k}$-algebra A and an ideal $I \subset A$, one defines the *associated graded algebra* as follows:

$$\mathrm{gr}_I(A) = A/I \oplus I/I^2 \oplus I^2/I^3 \oplus \cdots,$$

where:

(a) I^k is the *k-th power of I,* i.e. the ideal generated by all possible k-fold products $a_1 \cdots a_k$, where $a_1, \ldots, a_k \in I$,

(b) The multiplicative structure is determined by the pairings:

$$I^k/I^{k+1} \times I^l/I^{l+1} \to I^{k+l}/I^{k+l+1},$$

$$(\overline{a}, \overline{b}) \mapsto \overline{ab},$$

$$k, l \in \mathbb{Z}_+.$$

(Here we assume $I^0 = A$.)

The importance of this construction is that the affine scheme of $\mathrm{gr}_I(A)$ is a *flat deformation* of that of A, called the *Rees deformation* [Eis95, Section 6.4] and it plays an important role in resolutions of singularities in algebraic geometry.

Chapter 4

The main theorem

Throughout this section we assume $M \subset \mathbb{Z}_+$ is a numerical monoid, generated by coprime numbers $m_1, \ldots, m_e \in \mathbb{N}$, satisfying $m_1 < \cdots < m_e$. We follow the notation introduced in Sections 2.2.

4.1 Associated graded ring of a numerical monoid ring

Let $\mathbf{k}$ be a field. Denote by $\mathrm{gr}(\mathbf{k}[M])$ the associated graded ring with respect to the maximal monomial ideal $I \subset \mathbf{k}[M]$, i.e., the ideal, generated by $M \setminus \{1\}$. (Recall, in monoid rings we use multiplicative notation for the monoid operation.)

Proposition 4.1.

(a) We have the graded structure

$$\mathrm{gr}(\mathbf{k}[M]) \cong \mathbf{k} \oplus A_1 \oplus A_2 \oplus \cdots ,$$

making $\mathrm{gr}(\mathbf{k}[M])$ *into a homogeneous graded algebra, i.e.,* $\mathrm{gr}(\mathbf{k}[M])$ *is generated in degree one, i.e.,* $\mathrm{gr}(\mathbf{k}[M]) = \mathbf{k}[A_1]$*, where the homogeneous components are:*

$$A_k = \bigoplus_{\substack{m \in M \setminus \{1\} \\ l(m) = k}} \mathbf{k}m$$

(b) For every $m, n \in M \setminus \{1\}$*, their product* $m * n$ *in* $\mathrm{gr}(\mathbf{k}[M])$ *is given by*

$$m * n = \begin{cases} mn, \text{ if } \boldsymbol{l}(m) + \boldsymbol{l}(n) = \boldsymbol{l}(mn) \\ 0, \text{ if } \boldsymbol{l}(m) + \boldsymbol{l}(n) < \boldsymbol{l}(mn). \end{cases}$$

(c) The ring $\mathbf{k}[\mathbb{Z}_+ m_1]$ *(*$\cong \mathbf{k}[X]$*) is a graded* $\mathbf{k}$*-retract of* $\mathrm{gr}(\mathbf{k}[M])$*, i.e.,* $\mathbf{k}[\mathbb{Z}_+ m_1]$ *is a sub-algebra of* $\mathbf{k}[M]$ *and there is a* $\mathbf{k}$*-algebra homomorphism* $\mathbf{k}[M] \to \mathbf{k}[\mathbb{Z}_+ m_1]$*, which restricts to the identity map on* $\mathbf{k}[\mathbb{Z}_+ m_1]$*; moreover,* $\ker(\mathrm{gr}(\mathbf{k}[M]) \to \mathbf{k}[\mathbb{Z}_+ m_1])$ *is the nilradical of* $\mathrm{gr}(\mathbf{k}[M])$*.*

Proof. The parts (a, b) follow from the definition of $\mathrm{gr}(\mathbf{k}[M])$ and the equality $I^k \cap M = \{x_1 \cdots x_k | \ x_1, \ldots, x_k \in M\}$ for every $k \in \mathbb{N}$.

(c) That $\mathbf{k}[\mathbb{Z}_+ m_1]$ is a $\mathbf{k}$-vector subspace of $\operatorname{gr}(\mathbf{k}[M])$ is clear. That the multiplicative structures also agree follows from the part (b) and the observation that $\mathbf{l}(m_1^k) = k$ for every $k \in \mathbb{N}$.

We define the map $f : \operatorname{gr}(\mathbf{k}[M]) \to \mathbf{k}[\mathbb{Z}_+ m_1]$ to be the $\mathbf{k}$-algebra homomorphism, defined by

$$f(m) = \begin{cases} m \ \text{ if } \ m \in \mathbb{Z}_+ m_1, \\ \\ 0 \ \text{ if } \ m \in M \setminus \mathbb{Z}_+ m_1. \end{cases}$$

To see that f is well defined and $\ker(f) = \sqrt{0} \subset \operatorname{gr}(\mathbf{k}[M])$ we observe that every element $m \in M \setminus \mathbb{Z}_+ m_1$ is nilpotent in $\operatorname{gr}(\mathbf{k}[M])$. In fact, the degree of m, viewed as an element of $\operatorname{gr}(\mathbf{k}[M])$, is $\mathbf{l}(m) < \left\lfloor \frac{m}{m_1} \right\rfloor$. We claim $m^{*m_1} = 0 \in \operatorname{gr}(\mathbf{k}[M])$, where the star stands for the exponentiation in the algebra $\operatorname{gr}(\mathbf{k}[M])$. In fact, if $m^{*m_1} \in \operatorname{gr}(\mathbf{k}[M]) \setminus \{0\}$ then we can write

$$m > \deg(m^{*m_1}) = \deg(m_1^{*m}) = m,$$

a contradiction. $\square$

(For a similar description of $\operatorname{gr}(\mathbf{k}[N])$ for the *affine monoids* of high rank, see [Gub].)

Corollary 4.2.

(a) As a $\mathbf{k}$-vector space, $\operatorname{gr}(\mathbf{k}[M])$ can be thought of as the same $\mathbf{k}[M]$, where the product in $\operatorname{gr}(\mathbf{k}[M])$ is the modification of that in $\mathbf{k}[M]$ according to Proposition 4.1(b).

(b) The Hilbert function $H := H_{\operatorname{gr}(\mathbf{k}[M])}$ is given by $H(k) = d_k(M)$, $k \in \mathbb{N}$.

(c) The sequence $d_k(M)$ is eventually constant.

Proof. The part (b) is immediate from Proposition 4.1(a). The part (c) follows from Lemma 3.5 and Proposition 4.1(c). □

Example 4.3. Neither the sequence $d_k(M)$ nor even the ring $\mathrm{gr}(\mathbf{k}[M])$ itself determines the monoid M. In fact, for the numerical monoids $M_1 = \mathbb{Z}_+2 + \mathbb{Z}_+3$ and $M_2 = \mathbb{Z}_+2 + \mathbb{Z}_+5$, the algebras $\mathrm{gr}(\mathbf{k}[M_1])$ and $\mathrm{gr}(\mathbf{k}[M_2])$ are both graded isomorphic to $\mathbf{k}[X,\epsilon]/(\epsilon^2 = 0)$, where $\deg X = \deg \epsilon = 1$.

4.2 The second Frobenius number

In view of Corollary 4.2(c) we can introduce the following

Definition 4.4. The *second Frobenius number* of M, denoted by $\mathbf{F}'(M)$ is the smallest natural number k, such that $d_{k'}(M) = d_{k'+1}(M)$ for every $k' \geq k$.

Corollary 4.2(b) explains why this definition is analogous to the classical Frobenius number $\mathbf{F}(M)$: the latter can be defined as the smallest natural number k, such that $H_{\mathbf{k}[M]}(k') = H_{\mathbf{k}[M]}(k'+1)$, where $H_{\mathbf{k}[M]}$ is the Hilbert function of $\mathbf{k}[M]$ with respect of the standard grading $\deg(m) = m$ for the elements $m \in M$.

Our goal is to give an explicit upper bound for $\mathbf{F}'(M)$, not accessible via the theory of Hilbert functions of graded algebras.

Lemma 4.5. *For every element $(a_1, \dots, a_e) \in \boldsymbol{dec}(M)$, $a_i > 0$ implies $m_i \in \mathrm{Hilb}(M)$.*

Proof. This is equivalent to the claim that, for a maximal decomposition $m = \sum_{i=1}^{e} a_i m_i$ and an index $1 \leq i \leq e$, the inequality $a_i > 0$ implies $m_i \in \mathrm{Hilb}(M)$. In fact, if $m_i \notin \mathrm{Hilb}(M)$ then m_i is a nontrivial positive integer combination of elements of $\mathrm{Hilb}(M) \subset G$, contradicting the assumption that $\sum_{i=1}^{e} a_i m_i$ is a maximal decomposition. □

As a corollary the numbers $d_k(M)$ are independent of the choice of the generating set G of M: they only depend on $\mathrm{Hilb}(M)$.

Lemma 4.6. *Assume $(a_1, \dots, a_e) \in \boldsymbol{dec}(M)$ and $(b_1, \dots, b_e) \in \mathbb{Z}_+^e \setminus \{0\}$ satisfy $b_i \leq a_i$ for $i = 1, \dots, e$. Then $(b_1, \dots, b_e) \in \boldsymbol{dec}(M)$.*

Proof. This is equivalent to the claim that, for a maximal length decomposition $\sum_{i=1}^{e} a_i m_i$, any non-zero sub-sum $\sum_{i=1}^{e} b_i m_i$ (i.e., $b_i \leq a_i$ for every index i) is also a maximal length decomposition. In fact, if $\sum_{i=1}^{e} b_i m_i = \sum_{i=1}^{e} b'_i m_i$ for some $b'_i \in \mathbb{Z}_+$ with $\sum_{i=1}^{e} b_i < \sum_{i=1}^{e} b'_i$, then $\sum_{i=1}^{e} a_i m_i = \sum_{i=1}^{e} (a_i - b_i + b'_i) m_i$. Since $a_i - b_i + b'_i \in \mathbb{Z}_+$ and $\sum_{i=1}^{e} a_i < \sum_{i=1}^{e} (a_i - b_i + b'_i)$, this contradicts the assumption that $\sum_{i=1}^{e} a_i m_i$ is a maximal length decomposition. □

Theorem 4.7.

(a) *For every natural number $k \geq (e-1)m_1$, one has*

$$\#\boldsymbol{dec}_k(M) \geq \#\boldsymbol{dec}_{k+1}(M) \quad \text{and} \quad d_k(M) \geq d_{k+1}(M).$$

(b) For every natural number $k \geq (e-1)m_e$, one has

$$\#\boldsymbol{dec}_k(M) = \#\boldsymbol{dec}_{k+1}(M) \quad \textit{and} \quad d_k(M) = d_{k+1}(M).$$

In particular $\mathbf{F}'(M) \leq (e-1)m_e$.

(c) For every pair of natural numbers $a, b > (e-1)m_1m_e$, one has $a, b \in M$ and

$$\boldsymbol{l}(a) = \boldsymbol{l}(b) \quad \Longleftrightarrow \quad \left\lfloor \frac{a}{m_1} \right\rfloor = \left\lfloor \frac{b}{m_1} \right\rfloor.$$

In particular $d_k(M) = m_1$ for $k \geq (e-1)m_e$.

Remark. (a) Although it follows from Theorem 4.7(c) that $\mathbf{F}(M) \leq (e-1)m_1m_e$, a much better bound for $\mathbf{F}(M)$ is known, namely $\mathbf{F}(M) < m_1m_e$ (Section 2.3). On the other hand, the periodic behavior in Theorem 4.7(c) is more than an upper bound for $\mathbf{F}(M)$.

(b) It follows from [ONe17] that the function $\mathbf{l}(-)$ is eventually of the form in Theorem 4.7(c), but we also give an explicit lower bound from where this behavior shows up.

We will need two lemmas.

Lemma 4.8. *For every $(a_1, \ldots, a_e) \in \boldsymbol{dec}(M)$ and $i \in \{2, \ldots, e\}$, one has $a_i < m_1$.*

Proof. Assume to the contrary $a_j \geq m_1$ for some $j \geq 2$. Then we can write

$$a_1m_1 + \cdots + a_em_e =$$

$$a'_1m_1 + a_2m_2 + \cdots + a_{j-1}m_{j-1} + a'_jm_j + a_{j+1}m_{j+1} + \cdots + a_em_e,$$

where $a'_1 = a_1 + m_j$ and $a'_j = a_j - m_1$. This contradicts the containment $(a_1, \ldots, a_e) \in \mathbf{dec}(M)$ because $a_1 + a_j < a'_1 + a'_j$. $\square$

To state the second lemma, we first introduce the following objects:

(a) For every $k \in \mathbb{Z}_+$, the affine hyperplane and affine half-space:

$$\mathsf{G}_k := (X_1 + \cdots + X_e = k) \ \subset \ \mathbb{R}^e,$$

$$\mathsf{G}_k^- := (X_1 + \cdots + X_e \leq k) \ \subset \ \mathbb{R}^e;$$

(b) The two infinite right prisms:

$$\Pi_+ = \{(x_1, x_2, \ldots, x_e) \mid x_1, x_2, \ldots, x_e \geq 0 \ \text{ and } \\ x_2 + \cdots + x_e \leq (e-1)m_1\} \subset \mathbb{R}_+^e,$$

and

$$\Pi = \{(x_1, x_2, \ldots, x_e) \mid x_2, \ldots, x_e \geq 0 \ \text{ and } \\ x_2 + \cdots + x_e \leq (e-1)m_1\} \subset \mathbb{R} \times \mathbb{R}_+^{e-1};$$

(c) The linear map

$$\Gamma : \mathbb{R}^e \to \mathbb{R},$$
$$(x_1, \ldots, x_e) \mapsto \sum_{i=1}^{e} m_i x_i;$$

(d) The sequence of affine hyperplanes $\{\mathcal{H}_s\}_{i=0}^{\infty}$ in $\mathbb{R}^e$, defined by the following conditions:

  (a) each $\mathcal{H}_s$ is parallel to the hyperplane $\ker \Gamma$,

  (b) For every s, the set $\mathcal{H} \cap \mathbb{Z}_+^e$ is not empty,

(c) the distances δ_s between $\mathcal{H}_s$ and 0 form a strictly increasing sequence of non-negative real numbers,

(d) $\mathbb{Z}_+^e \subset \bigcup_{i=0}^{\infty} \mathcal{H}_s$;

(e) The sequence of lattice polytopes

$$P_s = \operatorname{conv}(\mathcal{H}_s \cap \Pi_+ \cap \mathbb{Z}^e) \subset \mathbb{R}_+^e, \qquad s = 0, 1, \dots$$

(for some initial vlues of s the polytope P_s may be empty).

For $s \in \mathbb{Z}_+$, the *s-th element of M* refers to the s-the element of M in the natural order.

Lemma 4.9.

(a) $\{\delta_s\}_{s=0}^{\infty}$ is additive submonoid of $\mathbb{R}_+$, the non-negative reals, and isomorphic to M;

(b) The set of presentations $m = \sum_{i=1}^{e} a_i m_i$ with $a_i \in \mathbb{Z}_+$ of the s-th element $m \in M$ is bijective to $\mathcal{H}_s \cap \mathbb{Z}_+^e$;

(c) For every $k \in \mathbb{N}$, $d_k(M)$ equals the number of those indices s, for which $P_s \cap \mathsf{G}_k \neq \emptyset$ and $P_s \subset \mathsf{G}_k^-$.

Proof. Part (a) follows from the definition of the $\mathcal{H}_s$, Part (b) is a consequence of (a), and Part (c) is a consequence of (b) in view of Lemma 4.8 and the inclusion

$$\{(a_1, \dots, a_e) \mid a_1, \dots, a_e \in \mathbb{Z}_+ \text{ and } a_2, \dots, a_e \leq m_1 - 1\} \ \subset \ \Pi_+.$$

$\square$

Notice. Instead of Π_+ and Π we could have chosen the narrower prisms, defined by the inequalities $x_2 + \cdots + x_e \le (e-1)(m_1 - 1)$. This would result in minor improvements of the bounds in Theorem 4.7(b, c), but at the expense of complicated quotient expressions for the lower bounds instead of natural numbers.

Proof of Theorem 4.7. (a) For a natural number $k > (e-1)m_1$ and an element $(a_1, \ldots, a_e) \in \mathbf{dec}_k(M)$, Lemma 4.8 implies $a_1 > 0$. Consequently, for every $k \ge (e-1)m_1$, Lemma 4.6 implies the injective map

$$\iota_{k+1} : \mathbf{dec}_{k+1}(M) \to \mathbf{dec}_k(M),$$
$$(a_1, a_2, \ldots, a_e) \mapsto (a_1 - 1, a_2, \ldots, a_e).$$

This proves the inequalities for $\#\mathbf{dec}_k(M)$.

For every $k \in \mathbb{N}$, we have $d_k(M) = \#\Gamma\big(\mathbf{dec}_k(M)\big)$. Consequently, the inequalities for $d_k(M)$ follow from the observation that, for an index $k \ge (e-1)m_1$ and elements $(a_1, \ldots, a_e), (b_1, \ldots, b_e) \in \mathbf{dec}_{k+1}(M)$, the following implication holds:

$$\Gamma\big((a_1, \ldots, a_e)\big) \ne \Gamma\big((b_1, \ldots, b_e)\big) \implies$$
$$\Gamma\big(\iota_{k+1}(a_1, \ldots, a_e)\big) \ne \Gamma\big(\iota_{k+1}(b_1, \ldots, b_e)\big).$$

(b) Fix a natural number $k \ge (e-1)m_e$. Since m_1 is the smallest generator of M we have $k\mathbf{e}_1 \in \mathbf{dec}_k(M)$. It represents the element $km_1 \in M$. Assume $ke_1 = \Gamma(\mathcal{H}_s \cap \mathbb{Z}_+^e)$ for some $s \in \mathbb{N}$, i.e., km_1 is the s-th element in M.

We claim that

$$\mathcal{H}_s \cap \Pi_+ = \mathcal{H}_s \cap \Pi. \tag{4.1}$$

This equality is equivalent to the condition that, for every $i \in \{2, \dots, e\}$, the first coordinate of the point

$$\mathcal{H}_s \cap \big((e-1)m_1\mathbf{e}_i + \mathbb{R}\mathbf{e}_1\big)$$

is non-negative, or equivalently, for every $i \in \{2, \dots, e\}$, the first coordinate of the point

$$\mathcal{H}_0 \cap \big((e-1)m_1\mathbf{e}_i + \mathbb{R}\mathbf{e}_1\big) = \big(-k\mathbf{e}_1 + \mathcal{H}_s\big) \cap \big((e-1)m_1\mathbf{e}_i + \mathbb{R}\mathbf{e}_1\big)$$

is at least $-(e-1)m_e$. The mentioned point is the solution to the equation

$$m_1x_1 + \dots + m_ex_e = 0,$$

subject to $x_j = 0$ for $j \neq 1, i$ and $x_i = (e-1)m_1$. The first coordinate of this points is $-(e-1)m_i \geq -(e-1)m_e$. This proves (4.1).

For every element $m \in M$ with $\mathbf{l}(m) = k$ we have $km_1 \leq m$, implying $m = \Gamma(\mathcal{H}_t \cap \mathbb{Z}^e_+)$ for some $t > s$. In particular, Lemma 4.9(c) implies that, for every index t with $P_t \cap \mathsf{G}_k \neq 0$ and $P_t \subset \mathsf{G}_k^-$, one has $t > s$ and, therefore, (4.1) implies $\mathcal{H}_t \cap \Pi = \mathcal{H}_t \cap \Pi_+$. This, in turn, implies the following.

Claim. The set of hyperplanes $\mathcal{H}_t$, satisfying $P_t \cap \mathsf{G}_{k+1} \neq 0$ and $P_t \subset \mathsf{G}_{k+1}^-$ is the parallel translate by $\mathbf{e}_1$ of the set of hyperplanes $\mathcal{H}_{t'}$, satisfying $P_{t'} \cap \mathsf{G}_k \neq 0$ and $P_{t'} \subset \mathsf{G}_k^-$.

Lemma 4.9 and the Claim together imply Part (b).

(c) Lemma 4.9 and the Claim above also imply that, starting from the $((e-1)m_e)$-th member, the monoid M exhibits a periodic pattern: denoting by μ this member, a natural number $n \geq \mu$ is in M if and only if $n + h \in M$, where

$$h = \frac{\Gamma(\mathbf{e}_1)}{\min(\Gamma(\mathbb{Z}^e) \cap \mathbb{N})} = \frac{m_1}{\gcd(m_1, \ldots, m_e)} = m_1. \tag{4.2}$$

Since $n \in M$ for $n \gg 0$, one has $\mathbf{F}(M) < \mu$. But, obviously, $\mu \leq (e-1)m_e m_1$.

As above, we assume $k\mathbf{e}_1 \in \mathcal{H}_s$. For every $t \geq s$, the equality (4.2) implies

$$\mathcal{H}_t \cap \mathbb{R}\mathbf{e}_1 = \frac{t-s+1}{m_1} \cdot \mathbf{e}_1. \tag{4.3}$$

For the equalities in Part (c), one needs to show that $P_t \cap \mathsf{G}_k \neq 0$ and $P_t \subset \mathsf{G}_k^-$ for every index $s < t \leq s + m_1 - 1$. Assume to the contrary this is not the case for some index t. Then, $P_t \cap \mathsf{G}_r \neq 0$ and $P_t \subset \mathsf{G}_r^-$ for some $r > k$. By Sublemma, $\mathcal{H}_t$ is the parallel translate by $(r-k)\mathbf{e}_1$ of some $\mathcal{H}_{t'}$ with $s \leq t' < t$. We arrive at the contradiction

$$\frac{t-t'}{m_1} = \frac{t-s+1}{m_1} - \frac{t'-s+1}{m_1} = r - k \geq 1,$$

where the second equality is due to (4.3). $\square$

Chapter 5

Generating the $d_k(M)$

We keep the notation, introduced in Section 2.2.

Given a natural number n, in this section we describe an algorithm for computing the numbers $d_1(M), d_2(M), \ldots, d_n(M)$.

5.1 Algorithm 1.

Step 1. Declare a struct called node to encapsulate a monoid element's value and length, which is not necessarily its maximal decomposition length. This node will be used in a linked list which is generated in the next step.

Step 2. To compute the monoid with generating set $\{m_1, \ldots, m_e\}$, create e for-loops where each loop ranges from 0 to $\frac{nm_e}{m_1}$. The nested loops will create duplicate elements with different coefficients and, not necessarily, different lengths (an element can have more than one representation with the same length).

It is important to create enough copies of each element less than $nm_e - m_1$ so that element's maximum decomposition is found. An element less than $nm_e - m_1$ can be generated with coefficients whose sum is less than or equal to n, but its maximum decomposition is not found.

During the loop, check if the element is less than $nm_e - m_1$ before creating a node and inserting into the linked list.

Step 3. Create an array of length $nm_e - m_1$. Hash the linked list into an array where each element of the array corresponds to an element in the linked list. Before inserting a node into the array, check if !NULL and if the new node has a length greater than the existing node. If NULL, insert regardless. Thus the elements of the array will either be NULL or contain a pointer to one node with the maximum decomposition length of that element.

```
LET m = monoid element

IF (m'th element of arr is NULL)

    ASSIGN node to array

    Set next to NULL

ELSE IF (length of node > length of node in array m'th element)

    FREE node in m'th position

    ASSIGN node to array

    Set next to NULL
```

Step 4. Create decomposition array of size n. Hash previous array into decomposition array by creating a linked list of nodes who all have a maximal decomposition length equal to the decomposition array's element position.

```
IF (element array is not NULL and max decomp length is less than n)
    ASSIGN node in decomposition array to new node next
    ASSIGN new node to decomposition array in length position in array
```

The number of nodes in the linked list in each position k of the decomposition array will equal $d_k(M)$. Count and print those lists and you are done.

5.2 Algorithm 2

Step 1. To reduce the computations, one can first extract $\mathrm{Hilb}(M)$ from the generating set $\{m_1, \ldots, m_e\}$. This can be done using Normaliz. The steps below are independent of this steps, though.

Step 2. Define $M_k = [1, k] \cap M$ and generate the ascending chain of sets

$$M_1 \subset M_2 \subset M_3 \subset \ldots \subset M_{nm_e - m_1},$$

inductively as follows. Without loss of generality we can assume $m_1 > 1$, for otherwise

$M = \mathbb{Z}_+$ and everything trivializes. Put $M_1 = \{0\}$ and, for $k > 1$,

$$M_k = \begin{cases} M_{k-1}, & \text{if } \big(k - \mathrm{Hilb}(M)\big) \cap M_{k-1} = \emptyset, \\ M_{k-1} \cup \{k\}, & \text{if } \big(k - \mathrm{Hilb}(M)\big) \cap M_{k-1} \neq \emptyset. \end{cases}$$

Step 3. We construct the subsets $\mathbf{dec}_k(M) \subset \mathbb{Z}_+^e$ inductively. Put

$$\mathbf{dec}_1(M) = \{\mathbf{e}_1, \dots, \mathbf{e}_e) \text{ and } d_1 = e.$$

Assume we have generated $\mathbf{dec}_k(M)$ for some $k \geq 1$. Then one generates $\mathbf{dec}_{k+1}(M)$ as follows.

By Lemma 4.6, we have the inclusion

$$\mathbf{dec}_{k+1}(M) \subset \bigcup_{i=1}^{e} \big(\mathbf{e}_i + \mathbf{dec}_k(M)\big)$$

For every element $x \in \bigcup_{i=1}^{e} \big(\mathbf{e}_i + \mathbf{dec}_k(M)\big)$, one verifies the inclusion $x \in \mathbf{dec}_{k+1}(M)$ by checking the following condition:

$$\big\{\Gamma(x) - \Gamma(y) \mid y \in \mathbf{dec}_k(M)\big\} \cap \big(M_{(k+1)m_e - km_1} \setminus \mathrm{Hilb}(M)\big) = \emptyset,$$

the map $\Gamma : \mathbb{Z}_+^e \to \mathbb{Z}_+$ as in Section 4.2.

Step 4. After generating the sets $\mathbf{dec}_1(M), \dots, \mathbf{dec}_n(M)$, the numbers $d_k(M)$ are determined by

$$d_k(M) = \#\left\{\Gamma(x) \mid x \in \mathbf{dec}_k(M)\right\}.$$

Chapter 6

Algorithms

6.1 Algorithm 1

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <time.h>
#include <string.h>

// Define the max element length
const int N = 101;

struct node {
    int element;
    int coef[5];
    int length;
    struct node *next;
};

void hash_monoid(struct node **arr, struct node *list, int M_MAX, int E);
void hash_element_arr(struct node **arr0, struct node **arr1, int M_MAX);
void free_list(struct node *list);
int size(struct node *ptr);
void bubble_sort(int *arr, int n);
int gcd(int a, int b);
int findGCD(int *arr, int n);
void print_gen_set(int *arr, int E);
void create_gen_set(int *arr, int max, int min, int E);
void free_dec(struct node **n);
struct node *create_monoid(int *arr, int M_MAX, int M, int E);
void free_hash(struct node **arr, int M_MAX);
void print_to_file(struct node **arr0, int *arr1, int E);

int main(void)
```

```
{
    // Declare the embedding dimension variable E
    int E;

    // Define the upper and lower bounds for the generators.
    int max = 300;
    int min = 50;

    // Prompt the user for the number of generators
    printf("Program to calculate d_k(M) for k = {0, ..., %i}. The generators will be between %i and %i inclusive.\n", N - 1, min, max);
    printf("Please enter in the number of generators: ");
    scanf("%i", &E);

    clock_t start, end;
    double cpu_time_used;
    start = clock();

    // Define an array to hold the E generators
    int gen_set[E];

    // Create the set of generators, check if they are coprime, and if they are not, divide by the gcd.
    create_gen_set(gen_set, max, min, E);

    // Print to screen the generators
    print_gen_set(gen_set, E);

    // Define the max element in the numerical monoid
    const int M_MAX = (N*gen_set[E - 1]) - gen_set[0] + 1;
    printf("M_MAX: %i\n", M_MAX);

    // Define the number of times to loop through each generator.
    // LOOP_MAX must be large enough so that numbers less than M_MAX have their max decomposition calculated.
    // E.G. If M = <2,3>, then M_MAX = 302. Notice 298 = 149*2 + 0*3. Thus 298 has length of 149. But that wont be
    // calculated if loop only goes through N. So LOOP_MAX = 151.
    const int LOOP_MAX = N*gen_set[E-1] / gen_set[0];
    printf("LOOP_MAX: %i\n", LOOP_MAX);

    // Build the monoid from 0 to M_MAX
    struct node *monoid = create_monoid(gen_set, M_MAX, LOOP_MAX, E);

    // Hash monoid by element length, only keeping max decomposition length.
    // The i'th element of the array contains a pointer to the node with the maximum decomposition of element i.
```

```
    struct node *element_arr[M_MAX];
    hash_monoid(element_arr, monoid, M_MAX, E);

    // Hash element_arr by length. The i'th element of the array contains a pointer to a linked list of nodes
    // who all have a length of i.
    struct node *length_arr[N];
    hash_element_arr(length_arr, element_arr, M_MAX);

    // Print k and d_k(M) to csv file
    print_to_file(length_arr, gen_set, E);

    free_dec(length_arr);
    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("Program took %f seconds to execute\n", cpu_time_used);
    return 0;
}

// Function to build the numerical monoid linked list
struct node *create_monoid(int *arr, int M_MAX, int M, int E)
{
    int i, j, k, l, m;
    struct node *n;
    struct node *list;
    list = NULL;
    if (E == 2){
        for (i = 0; i < M; i++){
            for (j = 0; j < M; j++){
                //if (i + j < N) {
                if (i*arr[0] + j*arr[1] < M_MAX){
                    n = malloc(sizeof(*n));
                    if (!n){
                        free_list(list);
                        printf("ERROR\n");
                    }
                    // memset(n, 0, sizeof(*n));
                    n -> element = i*arr[0] + j*arr[1];
                    n -> coef[0] = i;
                    n -> coef[1] = j;
                    n -> length = i + j;
                    n -> next = list;
                    list = n;
                }
            }
        }
    }
    else if (E == 3){
        for (i = 0; i < M; i++){
```

```
            for (j = 0; j < M; j++){
                for (k = 0; k < M; k++){
                    if (i*arr[0] + j*arr[1] + k*arr[2] < M_MAX){
                        n = malloc(sizeof(*n));
                        if (!n){
                            free_list(list);
                            printf("ERROR\n");
                        }
                        n -> element = i*arr[0] + j*arr[1] + k*arr[2];
                        n -> coef[0] = i;
                        n -> coef[1] = j;
                        n -> coef[2] = k;
                        n -> length = i + j + k;
                        n -> next = list;
                        list = n;
                    }
                }
            }
        }
    }
    else if (E == 4){
        for (i = 0; i < M; i++){
            for (j = 0; j < M; j++){
                for (k = 0; k < M; k++){
                    for (l = 0; l < M; l++){
                        if (i*arr[0] + j*arr[1] + k*arr[2] + l*arr[3] < M_MAX){
                            n = malloc(sizeof(*n));
                            if (!n){
                                free_list(list);
                                printf("ERROR\n");
                            }
                            n -> element = i*arr[0] + j*arr[1] + k*arr[2] + l*arr[3];
                            n -> coef[0] = i;
                            n -> coef[1] = j;
                            n -> coef[2] = k;
                            n -> coef[3] = l;
                            n -> length = i + j + k + l;
                            n -> next = list;
                            list = n;
                        }
                    }
                }
            }
        }
    }
    else if (E == 5){
        for (i = 0; i < M; i++){
```

```
            for (j = 0; j < M; j++){
                for (k = 0; k < M; k++){
                    for (l = 0; l < M; l++){
                        for (m = 0; m < M; m++){
                            if (i*arr[0] + j*arr[1] + k*arr[2] + l*arr[3] + m*arr[4] < M_MAX){
                                n = malloc(sizeof(*n));
                                if (!n){
                                    free_list(list);
                                    printf("ERROR\n");
                                }
                                n -> element = i*arr[0] + j*arr[1] + k*arr[2] + l*arr[3] + m*arr[4];
                                n -> coef[0] = i;
                                n -> coef[1] = j;
                                n -> coef[2] = k;
                                n -> coef[3] = l;
                                n -> coef[4] = m;
                                n -> length = i + j + k + l + m;
                                n -> next = list;
                                list = n;
                            }
                        }
                    }
                }
            }
        }
    }
    else{
        printf("Incorrect number of generators\n");
        printf("ERROR\n");
    }
    return list;
}

// hash_monoid(element_arr, monoid, M_MAX, int E);
void hash_monoid(struct node **arr, struct node *list, int M_MAX, int E)
{
    int i, x;
    struct node *tmp = NULL;
    for (i = 0; i < M_MAX; i++)
        arr[i] = NULL;
    while (list != NULL) {
        tmp = list;
        list = tmp -> next;
        x = tmp -> element;
        if (x < M_MAX && arr[x] == NULL) {
            arr[x] = tmp;
            tmp -> next = NULL;
```

```
        }
        else if (x < M_MAX && tmp -> length > arr[x] -> length) {
            free(arr[x]);
            arr[x] = tmp;
            tmp -> next = NULL;
        }
        else {
            free(tmp);
        }
    }
}



// hash_element_arr(length_arr, element_arr, M_MAX);
void hash_element_arr(struct node **arr0, struct node **arr1, int M_MAX)
{
    int i, x;
    struct node *tmp = NULL;
    for (i = 0; i < N; i++)
        arr0[i] = NULL;
    for (i = 0; i < M_MAX; i++){
        if (arr1[i] != NULL){
            tmp = arr1[i];
            x = tmp -> length;
            if (x < N){
                tmp -> next = arr0[x];
                arr0[x] = tmp;
            }
            else{
                free(tmp);
            }
        }
    }
}

int size(struct node *ptr)
{
    int counter = 0;
    struct node *tmp = ptr;
    while (tmp != NULL){
        counter++;
        tmp = tmp -> next;
    }
    return counter;
}

void free_list(struct node *list)
{
```

```
    struct node *tmp;
    while (list != NULL){
        tmp = list -> next;
        free(list);
        list = tmp;
    }
}

void free_dec(struct node **n)
{
    struct node *tmp1, *tmp2;
    for (int i = 0; i < N; i++){
        tmp1 = n[i];
        while (tmp1 != NULL){
            tmp2 = tmp1 -> next;
            free (tmp1);
            tmp1 = tmp2;
        }
    }
}

void bubble_sort(int *arr, int n){
    int i = 0, j = 0, tmp;
    for (i = 0; i < n; i++){   // loop n times - 1 per element
        for (j = 0; j < n - i - 1; j++){ // last i elements are sorted
    ↪ already
            if (arr[j] > arr[j + 1]){  // swap if order is broken
                tmp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = tmp;
            }
        }
    }
}


// Function to return gcd of a and b
int gcd(int a, int b)
{
    if (a == 0)
        return b;
    return gcd(b % a, a);
}

// Function to find gcd of array of
// numbers
int findGCD(int *arr, int n)
{
    int result = arr[0];
```

```
    for (int i = 1; i < n; i++){
        result = gcd(arr[i], result);
        if(result == 1){
            return 1;
        }
    }
    return result;
}

void create_gen_set(int *arr, int max, int min, int E)
{
    int i;
    // Fill gen_set with random integers between min and max
    srand(time(NULL));
    for (i = 0; i < E; i ++){
        arr[i] = (rand() % (max - min)) + min;
    }
    // Sort gen_set from minimum value to maximum for readability
    bubble_sort(arr, E);
    // Check with the elements in gen_set are coprime.
    // If they are not, divide all elements by the gcd to get a coprime
    ↪ list.
    int gcd = findGCD(arr, E);
    if (gcd != 1){
        for (i = 0; i < E; i++){
            arr[i] = arr[i] / gcd;
        }
    }
}

void print_gen_set(int *arr, int E)
{
    printf("The generators are: ");
    for (int i = 0; i < E; i ++){
        printf("%i", arr[i]);
        if (i < E - 1){
            printf(", ");
        }
    }
    printf("\n");
}

void free_hash(struct node **arr, int M_MAX)
{
    struct node *tmp0, *tmp1;
    for (int i = 0; i < M_MAX; i++){
        tmp0 = arr[i];
        while (tmp0 != NULL){
            tmp1 = tmp0 -> next;
```

```
                free(tmp0);
                tmp0 = tmp1;
            }
        }
}

void print_to_file(struct node **arr0, int *arr1, int E)
{
    int i;
    char buffer[30];
    sprintf(buffer, "output/lengths_%i.csv", E);
    FILE *file = fopen(buffer, "w");
    if (file == NULL){
        printf("Could not open file\n");
    }
    fprintf(file, "k, d_k(M),");
    for (i = 0; i < E; i++){
        fprintf(file, "%i", arr1[i]);
        if (i < E - 1) {
            fprintf(file, ",");
        }
    }
    fprintf(file, "\n");
    for (i = 0; i < N; i++)
        fprintf(file, "%i, %i\n", i, size(arr0[i]));
    fclose(file);
}
```

6.2 Algorithm 2

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <time.h>
#include <assert.h>
#include <string.h>

// Define the max element length
const int N = 101;

struct node{
    // Define the numerical monoid element
    int element;
    struct node *next;
};

// Define a node for each element in dec_k (M) - the set of coefficients for each maximal decomposition of length k.
struct dec_node{
    // Define an array to hold the coefficients for each monoid element
    int coef[6];
    struct dec_node *next;
};

struct dec_node *create_dec_nodes(struct dec_node *n0, struct dec_node *n1, int *arr0, struct node **arr1, int E, int k);
struct node *create_monoid(struct node *n0, int x, int *arr0, int E);
bool compare(int *x, struct node *n, int E);
void print_MONOID(struct node **n, int M_MAX);
void print_dec_node_list(struct dec_node *n, int E);
void free_MONOID(struct node **n, int M_MAX);
void free_dec(struct dec_node **n);
void print_dec_node(const char *label, struct dec_node *n, int E);
void duplicates(struct dec_node *ptr, int E);
struct dec_node *remove_non_max(struct dec_node *n0, struct dec_node *n1, int *arr0, struct node **arr1, int E, int k);
bool find(int z, struct node *ptr, int *arr, int E);
int gamma(struct dec_node *n, int *arr, int E);
int size(struct dec_node *ptr);
int size_hash(struct node *ptr);
struct node *hash_dec_nodes(struct dec_node *n, int *arr0, int E);
void duplicates_hash(struct node *ptr, int E);
void free_hash(struct node **n);
void bubble_sort(int *arr, int n);
int gcd(int a, int b);
int findGCD(int *arr, int n);
void print_gen_set(int *arr, int E);
```

```
void create_gen_set(int *arr, int max, int min, int E);

int main(void)
{
    // Declare the embedding dimension variable E
    int i, E;

    // Define the upper and lower bounds for the generators.
    int max = 300;
    int min = 50;

    // Prompt the user for the number of generators
    printf("Program to calculate d_k(M) for k = {0, ..., %i}. The generators will be between %i and %i inclusive.\n", N - 1, min, max);
    printf("Please enter in the number of generators: ");
    scanf("%i", &E);

    // Define an array to hold the n generators
    int gen_set[E];

    // Create the set of generators, check if they are coprime,
    // and if they are not, divide by the gcd.
    create_gen_set(gen_set, max, min, E);

    // Print out what the random generators are
    print_gen_set(gen_set, E);

    // Start programming timming
    clock_t start, end;
    double cpu_time_used;
    start = clock();

    // Define the max element in the numerical monoid
    const int M_MAX = (N*gen_set[E - 1]) - gen_set[0];

    // Declare an array of pointers to monoid elements in struct node and fill with NULL
    struct node *MONOID[M_MAX];
    for (i = 0; i < M_MAX; i++)
        MONOID[i] = NULL;

    // Inductively create the monoid
    for (int i = 0; i < M_MAX; i++){
        if (i == 0){
            MONOID[0] = malloc(sizeof(struct node));
            if (!MONOID[0]) {
                printf("ERROR: line number %d in function %s\n", __LINE__, __func__);
```

```
            }
            memset(MONOID[0], 0, sizeof(*MONOID[0]));
        }
        else{
            MONOID[i] = create_monoid(MONOID[i - 1], i, gen_set, E);
        }
    }

    // Declare an array of pointers to a dec_node linked list and fill
    ↪ with NULL
    struct dec_node *dec[N];
    for (i = 0; i < N; i++)
        dec[i] = NULL;

    // Build dec_k(M)
    for (i = 0; i < N; i++){
        if (i == 0)
            dec[0] = create_dec_nodes(NULL, NULL, NULL, NULL, E, 0);
        else if (i == 1)
            dec[1] = create_dec_nodes(dec[0], NULL, NULL, NULL, E, 1);
        else
            dec[i] = create_dec_nodes(dec[1], dec[i-1], gen_set, MONOID, E
    ↪ , i);
    }

    // Remove duplicate elements with equal lengths
    struct node *hash_table[N];
    for (i = 0; i < N; i++)
        hash_table[i] = NULL;
    for (i = 0; i < N; i++)
        hash_table[i] = hash_dec_nodes(dec[i], gen_set, E);

    char buffer[23];
    sprintf(buffer, "output/lengths_%i.csv", E);

    FILE *file = fopen(buffer, "w");
    if (file == NULL){
        printf("Could not open file\n");
        return 1;
    }
    fprintf(file, "k, d_k(M),");
    for (i = 0; i < E; i++) {
        fprintf(file, "%i", gen_set[i]);
        if (i < E - 1) {
            fprintf(file, ",");
        }
    }
    fprintf(file, "\n");
    for (i = 0; i < N; i++)
```

```c
        fprintf(file, "%i, %i\n", i, size_hash(hash_table[i]));
    fclose(file);

    free_MONOID(MONOID, M_MAX);
    free_dec(dec);
    free_hash(hash_table);

    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;

    printf("Program took %f seconds to execute\n", cpu_time_used);
    return 0;
}

void create_gen_set(int *arr, int max, int min, int E)
{
    int i;
    // Fill gen_set with random integers between min and max
    srand(time(NULL));
    for (i = 0; i < E; i ++){
        arr[i] = (rand() % (max - min)) + min;
    }

    // Sort gen_set from minimum value to maximum for readability
    bubble_sort(arr, E);

    // Check with the elements in gen_set are coprime.
    // If they are not, divide all elements by the gcd to get a coprime list.
    int gcd = findGCD(arr, E);
    if (gcd != 1){
        for (i = 0; i < E; i++){
            arr[i] = arr[i] / gcd;
        }
    }
}

void print_gen_set(int *arr, int E)
{
    printf("The generators are: ");
    for (int i = 0; i < E; i ++){
        printf("%i", arr[i]);
        if (i < E - 1){
            printf(", ");
        }
    }
    printf("\n");
}
```

```

struct dec_node *create_dec_nodes(struct dec_node *n0, struct dec_node *n1
    ↪ , int *arr0, struct node **arr1, int E, int k)
{
    int i, j;
    struct dec_node *n;
    struct dec_node *list = NULL;
    struct dec_node *tmp1;

    if (n0 == NULL){
        list = malloc(sizeof(*list));
        if (!list) {
            printf("ERROR: line number %d in function %s\n", __LINE__,
    ↪ __func__);
        }
        memset(list, 0, sizeof(*list));
    }
    else if (n1 == NULL){
        for (i = 0; i < E; i++){ // Put the right number of zeros at the
    ↪ front of the vector
            n = malloc(sizeof(*n));
            if (!n) {
                printf("ERROR: line number %d in function %s\n", __LINE__,
    ↪  __func__);
            }
            for (j = 0; j < i; j++)
                n -> coef[j] = 0;
            n -> coef[j] = 1; // Put 1 in the right place
            for (j = j + 1; j < E; j++) // Fill in the rest of the zeros
                n -> coef[j] = 0;
            n -> next = list;
            list = n;
        }
    }
    else{
        while (n0 != NULL){
            tmp1 = n1;
            while (tmp1 != NULL){
                n = malloc(sizeof(*n));
                if (!n) {
                    printf("ERROR: line number %d in function %s\n",
    ↪ __LINE__, __func__);
                }
                for (i = 0; i < E; i++)
                    n -> coef[i] = n0 -> coef[i] + tmp1 -> coef[i];
                n -> next = list;
                list = n;
                tmp1 = tmp1 -> next;
            }
```

```c
            n0 = n0 -> next;
        }
        if (list != NULL) {
            duplicates(list, E);
            list = remove_non_max(list, n1, arr0, arr1, E, k);
        }
    }
    return list;
}

struct dec_node *remove_non_max(struct dec_node *n0, struct dec_node *n1, int *arr0, struct node **arr1, int E, int k)
{
    struct dec_node *tmp0, *tmp1, *prev;
    int M = (k+1)*arr0[E - 1] - k*arr0[0];
    tmp0 = n0;
    tmp1 = n1;

    // If n0 node itself holds the key to be deleted
    while (tmp1 != NULL && tmp0 != NULL){
        if (find(gamma(tmp0, arr0, E) - gamma(tmp1, arr0, E), arr1[M+1], arr0, E)){
            n0 = tmp0 -> next;
            free(tmp0);
            tmp0 = n0;
            tmp1 = n1;
        }
        else{
            tmp1 = tmp1 -> next;
        }
    }

    // Advance tmp0 since the first node is checked
    tmp0 = n0 -> next;
    prev = n0;

    while (tmp0 != NULL){
        tmp1 = n1;
        while (tmp1 != NULL){
            if (find(gamma(tmp0, arr0, E) - gamma(tmp1, arr0, E), arr1[M+1], arr0, E)){
                prev -> next = tmp0 -> next;
                free(tmp0);
                tmp0 = prev -> next;
                tmp1 = n1;
            }
            else{
                tmp1 = tmp1 -> next;
            }
```

```
        }
        if (tmp0 != NULL){
            tmp0 = tmp0 -> next;
            prev = prev -> next;
        }
    }
    return n0;
}

struct node *hash_dec_nodes(struct dec_node *n0, int *arr0, int E)
{
    struct dec_node *tmp = n0;
    struct node *list = NULL;
    struct node *n = NULL;
    while (tmp != NULL){
        n = malloc(sizeof(*n));
        if (!n) {
            printf("ERROR: line number %d in function %s\n", __LINE__,
    ↪ __func__);
        }
        n -> element = gamma(tmp, arr0, E);
        n -> next = list;
        list = n;
        tmp = tmp -> next;
    }
    duplicates_hash(list, E);
    return list;
}

void duplicates_hash(struct node *list, int E)
{
    struct node *tmp0, *tmp1, *prev;
    int i, j;
    tmp0 = list;
    while (tmp0 != NULL && tmp0 -> next != NULL){
        prev = tmp0;
        tmp1 = prev -> next;
        while (tmp1 != NULL){
            if (tmp0 -> element != tmp1 -> element) {
                prev = tmp1;
                tmp1 = prev -> next;
            }
            else if (tmp0 -> element == tmp1 -> element) {
                prev -> next = tmp1 -> next;
                free(tmp1);
                tmp1 = prev -> next;
            }
        }
        tmp0 = tmp0 -> next;
```

```c
    }
}

int gamma(struct dec_node *n, int *arr, int E)
{
    int x = 0;
    for (int i = 0; i < E && n != NULL; i++){
        x += n -> coef[i] * arr[i];
    }
    return x;
}

bool find(int z, struct node *ptr, int *arr, int E)
{
    // Check to see if z is an element in the Hilbert basis. If so, then return false.
    for (int i = 0; i < E; i++){
        if (z == arr[i])
            return false;
    }
    struct node *tmp_1 = ptr;
    while (tmp_1 != NULL){
        if (z == tmp_1 -> element){
            return true;
        }
        tmp_1 = tmp_1 -> next;
    }
    return false;
}

void print_dec_node_list(struct dec_node *n, int E)
{
    // Print out the elements in the i'th dec set
    while (n != NULL){
        printf("(%i", n -> coef[0]);
        for (int i = 1; i < E; i++)
            printf(", %i", n -> coef[i]);
        printf(")  ");
        n = n -> next;
    }
    printf("\n");
}

void print_dec_node(const char *label, struct dec_node *n, int E)
{
    printf("%s: ", label);
    printf("(%i", n -> coef[0]);
    for (int i = 1; i < E; i++)
        printf(", %i", n -> coef[i]);
```

```
    printf(")\n");
}

struct node *create_monoid(struct node *n0, int k, int *arr0, int E)
{
    int i;
    int arr1[E];
    struct node *tmp0 = n0;
    struct node *list = NULL;
    struct node *n;

    // Calculate the set k - Hilb(M)
    for (i = 0; i < E; i++){
        // *(arr1 + i) = k - *(arr0 + i);
        arr1[i] = k - arr0[i];
    }

    while (tmp0 != NULL)
    {
        n = malloc(sizeof(*n));
        if (!n){
            printf("ERROR: line number %d in function %s\n", __LINE__, __func__);
        }
        n -> element = tmp0 -> element;
        n -> next = list;
        list = n;
        tmp0 = tmp0 -> next;
    }
    if (compare(arr1, n0, E))
    {
        n = malloc(sizeof(*n));
        if (!n){
            printf("ERROR: line number %d in function %s\n", __LINE__, __func__);
        }
        n -> element = k;
        n -> next = list;
        list = n;
    }
    return list;
}

bool compare(int *x, struct node *n, int E)
{
    int i;
    while (n != NULL){
        for (i = 0; i < E; i++){
            if (*(x + i) == (*n).element)
```

```
                return true;
        }
        n = (*n).next;
    }
    return false;
}

void free_MONOID(struct node **n, int M_MAX)
{
    struct node *tmp1, *tmp2;

    for (int i = 0; i < M_MAX; i++){
        tmp1 = *(n + i);
        while (tmp1 != NULL){
            tmp2 = tmp1 -> next;
            free(tmp1);
            tmp1 = tmp2;
        }
    }
}

void free_dec(struct dec_node **n)
{
    struct dec_node *tmp1, *tmp2;

    for (int i = 0; i < N; i++){
        tmp1 = n[i];
        while (tmp1 != NULL){
            tmp2 = tmp1 -> next;
            free (tmp1);
            tmp1 = tmp2;
        }
    }
}

void free_hash(struct node **n)
{
    struct node *tmp1, *tmp2;

    for (int i = 0; i < N; i++){
        tmp1 = n[i];
        while (tmp1 != NULL){
            tmp2 = tmp1 -> next;
            free (tmp1);
            tmp1 = tmp2;
        }
    }
}
```

```
void print_MONOID(struct node **n, int M_MAX)
{
    struct node *tmp0;

    for (int i = 0; i < M_MAX; i++){
        tmp0 = *(n + i);
        while (tmp0 != NULL){
            printf("%i ", tmp0 -> element);
            tmp0 = tmp0 -> next;
        }
        printf("\n");
    }

}

void duplicates(struct dec_node *ptr, int E)
{
    struct dec_node *tmp0, *tmp1, *prev;

    int i, j;
    tmp0 = prev = ptr;
    while (tmp0 -> next != NULL){
        prev = tmp0;
        tmp1 = tmp0 -> next;
        while (tmp1 != NULL){
            //print_nodes("compare", tmp0, tmp1);
            for (j = 0; j < E; j++) {
                 if (tmp0 -> coef[j] != tmp1 -> coef[j]) break;
            }
            if (j == E) {
                /* duplicate */
                prev->next = tmp1->next;
                assert(tmp1 != ptr);
                free(tmp1);
                tmp1 = prev->next;
            }
            else{
                prev = tmp1;
                tmp1 = tmp1->next;
            }
        }
        tmp0 = tmp0 -> next;
    }
}

int size(struct dec_node *ptr)
{
    int counter = 0;
    struct dec_node *tmp = ptr;
```

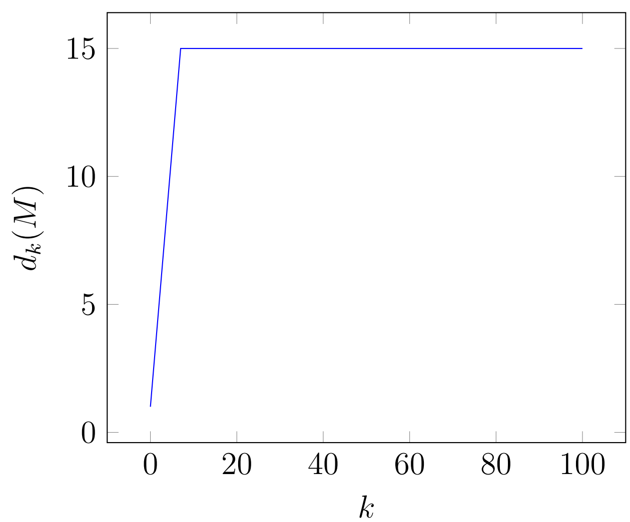
```
    while (tmp != NULL){
        counter++;
        tmp = tmp -> next;
    }
    return counter;
}

int size_hash(struct node *ptr)
{
    int counter = 0;
    struct node *tmp = ptr;
    while (tmp != NULL){
        counter++;
        tmp = tmp -> next;
    }
    return counter;
}

void bubble_sort(int *arr, int n) {
    int i = 0, j = 0, tmp;
    for (i = 0; i < n; i++){   // loop n times - 1 per element
        for (j = 0; j < n - i - 1; j++){ // last i elements are sorted
        ↪ already
            if (arr[j] > arr[j + 1]){  // swop if order is broken
                tmp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = tmp;
            }
        }
    }
}

// Function to return gcd of a and b
int gcd(int a, int b)
{
    if (a == 0)
        return b;
    return gcd(b % a, a);
}

// Function to find gcd of array of
// numbers
int findGCD(int *arr, int n)
{
    int result = arr[0];
    for (int i = 1; i < n; i++){
        result = gcd(arr[i], result);

        if(result == 1)
```

```
            return 1;
    }
    return result;
}
```

Chapter 7

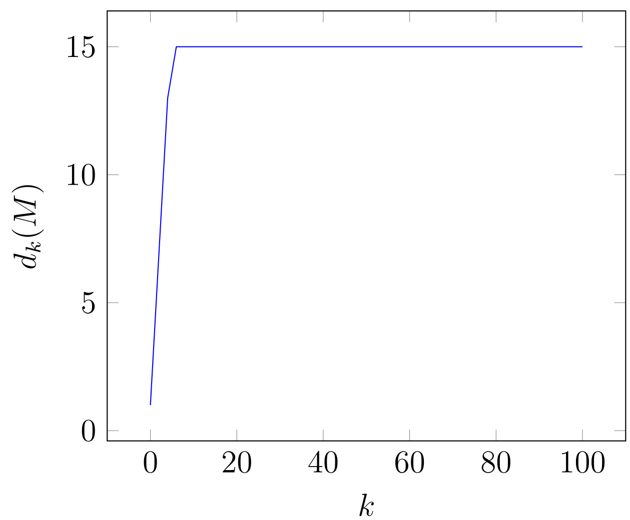
Computations

1. Numerical monoid generated by 15, 22, and 29



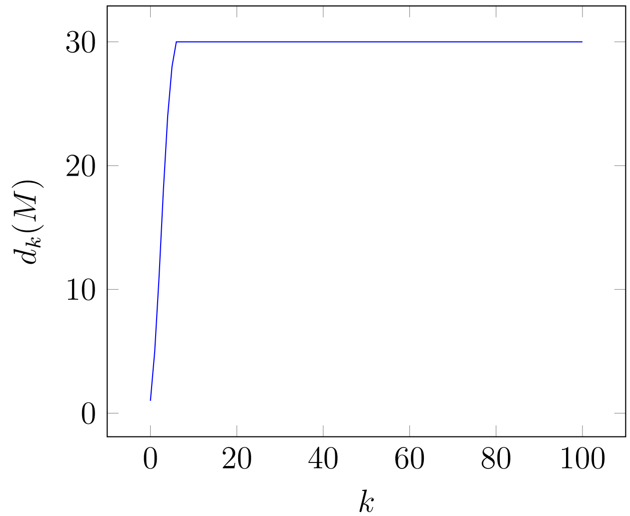
k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$
0	1	17	15	34	15	51	15	68	15	85	15
1	3	18	15	35	15	52	15	69	15	86	15
2	5	19	15	36	15	53	15	70	15	87	15
3	7	20	15	37	15	54	15	71	15	88	15
4	9	21	15	38	15	55	15	72	15	89	15
5	11	22	15	39	15	56	15	73	15	90	15
6	13	23	15	40	15	57	15	74	15	91	15
7	15	24	15	41	15	58	15	75	15	92	15
8	15	25	15	42	15	59	15	76	15	93	15
9	15	26	15	43	15	60	15	77	15	94	15
10	15	27	15	44	15	61	15	78	15	95	15
11	15	28	15	45	15	62	15	79	15	96	15
12	15	29	15	46	15	63	15	80	15	97	15
13	15	30	15	47	15	64	15	81	15	98	15
14	15	31	15	48	15	65	15	82	15	99	15
15	15	32	15	49	15	66	15	83	15	100	15
16	15	33	15	50	15	67	15	84	15		

2. Numerical monoid generated by 15, 22, 23 and 29



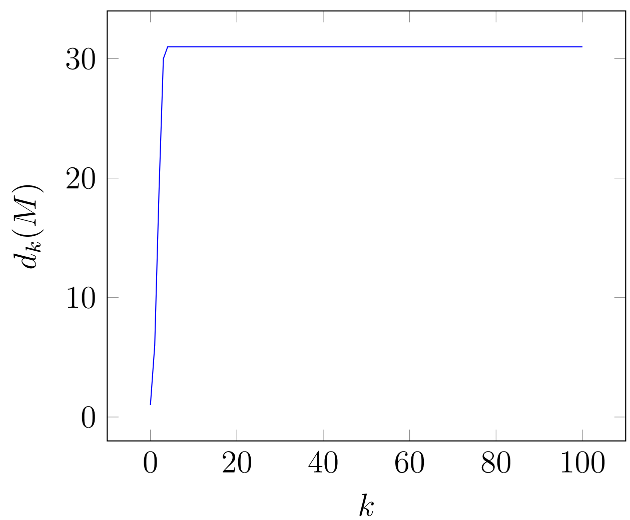
k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$
0	1	17	15	34	15	51	15	68	15	85	15
1	4	18	15	35	15	52	15	69	15	86	15
2	7	19	15	36	15	53	15	70	15	87	15
3	10	20	15	37	15	54	15	71	15	88	15
4	13	21	15	38	15	55	15	72	15	89	15
5	14	22	15	39	15	56	15	73	15	90	15
6	15	23	15	40	15	57	15	74	15	91	15
7	15	24	15	41	15	58	15	75	15	92	15
8	15	25	15	42	15	59	15	76	15	93	15
9	15	26	15	43	15	60	15	77	15	94	15
10	15	27	15	44	15	61	15	78	15	95	15
11	15	28	15	45	15	62	15	79	15	96	15
12	15	29	15	46	15	63	15	80	15	97	15
13	15	30	15	47	15	64	15	81	15	98	15
14	15	31	15	48	15	65	15	82	15	99	15
15	15	32	15	49	15	66	15	83	15	100	15
16	15	33	15	50	15	67	15	84	15		

3. Numerical monoid generated by 30, 35, 44, 46 and 58



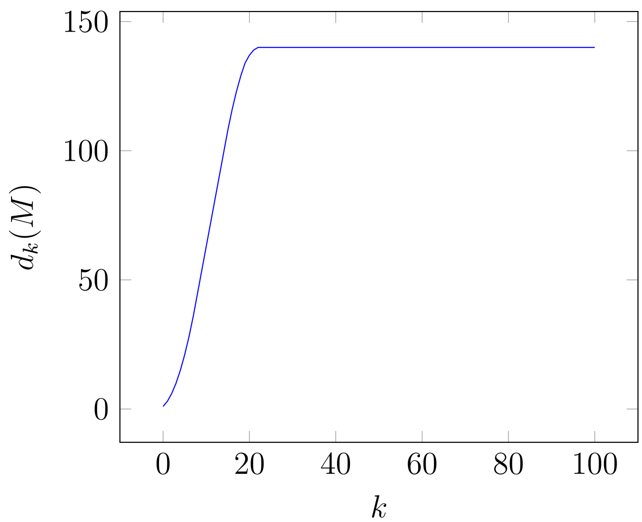
k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$
0	1	17	30	34	30	51	30	68	30	85	30
1	5	18	30	35	30	52	30	69	30	86	30
2	11	19	30	36	30	53	30	70	30	87	30
3	18	20	30	37	30	54	30	71	30	88	30
4	24	21	30	38	30	55	30	72	30	89	30
5	28	22	30	39	30	56	30	73	30	90	30
6	30	23	30	40	30	57	30	74	30	91	30
7	30	24	30	41	30	58	30	75	30	92	30
8	30	25	30	42	30	59	30	76	30	93	30
9	30	26	30	43	30	60	30	77	30	94	30
10	30	27	30	44	30	61	30	78	30	95	30
11	30	28	30	45	30	62	30	79	30	96	30
12	30	29	30	46	30	63	30	80	30	97	30
13	30	30	30	47	30	64	30	81	30	98	30
14	30	31	30	48	30	65	30	82	30	99	30
15	30	32	30	49	30	66	30	83	30	100	30
16	30	33	30	50	30	67	30	84	30		

4. Numerical monoid generated by 31, 33, 37, 38, 47 and 51



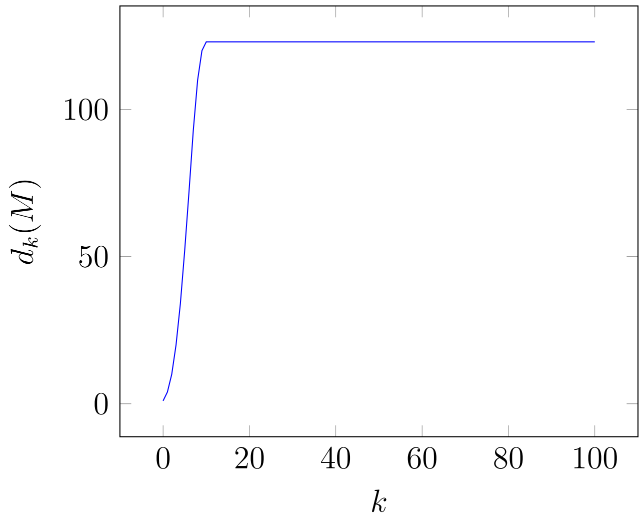
k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$
0	1	17	31	34	31	51	31	68	31	85	31
1	6	18	31	35	31	52	31	69	31	86	31
2	19	19	31	36	31	53	31	70	31	87	31
3	30	20	31	37	31	54	31	71	31	88	31
4	31	21	31	38	31	55	31	72	31	89	31
5	31	22	31	39	31	56	31	73	31	90	31
6	31	23	31	40	31	57	31	74	31	91	31
7	31	24	31	41	31	58	31	75	31	92	31
8	31	25	31	42	31	59	31	76	31	93	31
9	31	26	31	43	31	60	31	77	31	94	31
10	31	27	31	44	31	61	31	78	31	95	31
11	31	28	31	45	31	62	31	79	31	96	31
12	31	29	31	46	31	63	31	80	31	97	31
13	31	30	31	47	31	64	31	81	31	98	31
14	31	31	31	48	31	65	31	82	31	99	31
15	31	32	31	49	31	66	31	83	31	100	31
16	31	33	31	50	31	67	31	84	31		

5. Numerical monoid generated by 140, 145 and 149



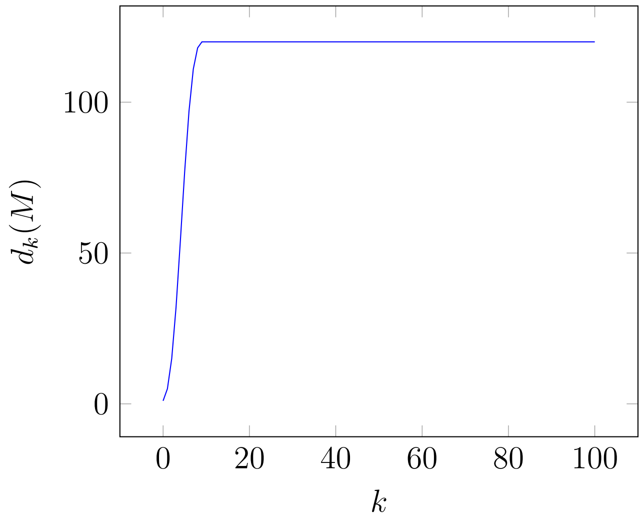
k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$
0	1	17	123	34	140	51	140	68	140	85	140
1	3	18	129	35	140	52	140	69	140	86	140
2	6	19	134	36	140	53	140	70	140	87	140
3	10	20	137	37	140	54	140	71	140	88	140
4	15	21	139	38	140	55	140	72	140	89	140
5	21	22	140	39	140	56	140	73	140	90	140
6	28	23	140	40	140	57	140	74	140	91	140
7	36	24	140	41	140	58	140	75	140	92	140
8	45	25	140	42	140	59	140	76	140	93	140
9	54	26	140	43	140	60	140	77	140	94	140
10	63	27	140	44	140	61	140	78	140	95	140
11	72	28	140	45	140	62	140	79	140	96	140
12	81	29	140	46	140	63	140	80	140	97	140
13	90	30	140	47	140	64	140	81	140	98	140
14	99	31	140	48	140	65	140	82	140	99	140
15	108	32	140	49	140	66	140	83	140	100	140
16	116	33	140	50	140	67	140	84	140		

6. Numerical monoid generated by 123, 126, 133 and 149



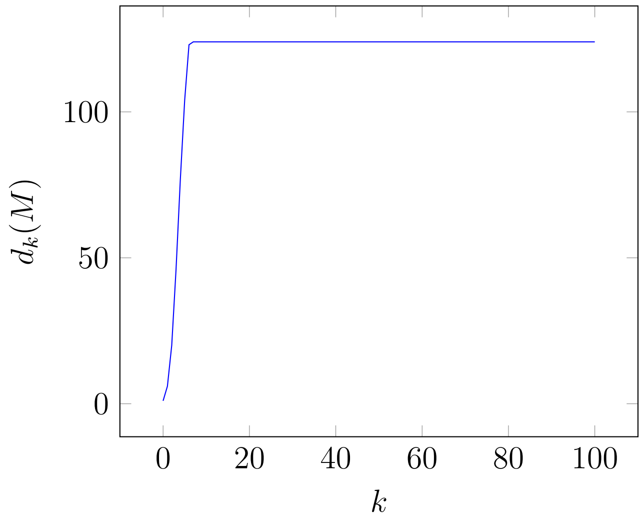
k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$
0	1	17	123	34	123	51	123	68	123	85	123
1	4	18	123	35	123	52	123	69	123	86	123
2	10	19	123	36	123	53	123	70	123	87	123
3	20	20	123	37	123	54	123	71	123	88	123
4	34	21	123	38	123	55	123	72	123	89	123
5	52	22	123	39	123	56	123	73	123	90	123
6	72	23	123	40	123	57	123	74	123	91	123
7	93	24	123	41	123	58	123	75	123	92	123
8	110	25	123	42	123	59	123	76	123	93	123
9	120	26	123	43	123	60	123	77	123	94	123
10	123	27	123	44	123	61	123	78	123	95	123
11	123	28	123	45	123	62	123	79	123	96	123
12	123	29	123	46	123	63	123	80	123	97	123
13	123	30	123	47	123	64	123	81	123	98	123
14	123	31	123	48	123	65	123	82	123	99	123
15	123	32	123	49	123	66	123	83	123	100	123
16	123	33	123	50	123	67	123	84	123		

7. Numerical monoid generated by 120, 126, 138, 139 and 141



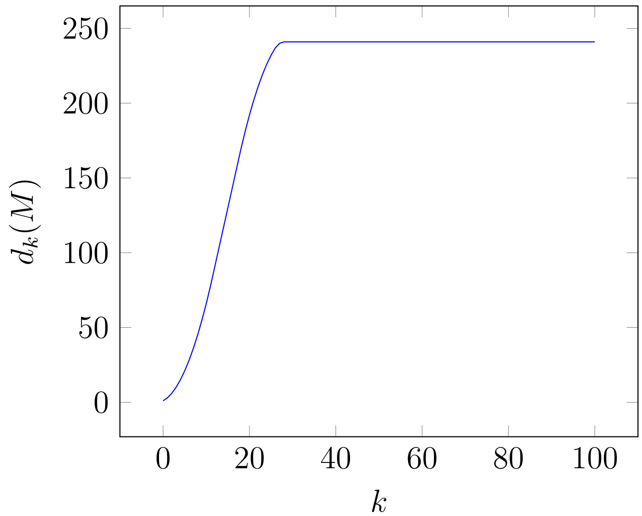
k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$
0	1	17	120	34	120	51	120	68	120	85	120
1	5	18	120	35	120	52	120	69	120	86	120
2	15	19	120	36	120	53	120	70	120	87	120
3	32	20	120	37	120	54	120	71	120	88	120
4	54	21	120	38	120	55	120	72	120	89	120
5	77	22	120	39	120	56	120	73	120	90	120
6	97	23	120	40	120	57	120	74	120	91	120
7	111	24	120	41	120	58	120	75	120	92	120
8	118	25	120	42	120	59	120	76	120	93	120
9	120	26	120	43	120	60	120	77	120	94	120
10	120	27	120	44	120	61	120	78	120	95	120
11	120	28	120	45	120	62	120	79	120	96	120
12	120	29	120	46	120	63	120	80	120	97	120
13	120	30	120	47	120	64	120	81	120	98	120
14	120	31	120	48	120	65	120	82	120	99	120
15	120	32	120	49	120	66	120	83	120	100	120
16	120	33	120	50	120	67	120	84	120		

8. Numerical monoid generated by 124, 127, 128, 135, 145 and 148



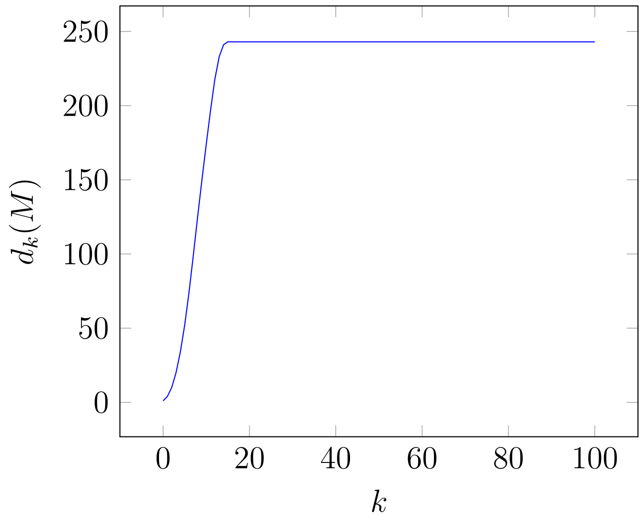
k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$
0	1	17	124	34	124	51	124	68	124	85	124
1	6	18	124	35	124	52	124	69	124	86	124
2	20	19	124	36	124	53	124	70	124	87	124
3	46	20	124	37	124	54	124	71	124	88	124
4	77	21	124	38	124	55	124	72	124	89	124
5	104	22	124	39	124	56	124	73	124	90	124
6	123	23	124	40	124	57	124	74	124	91	124
7	124	24	124	41	124	58	124	75	124	92	124
8	124	25	124	42	124	59	124	76	124	93	124
9	124	26	124	43	124	60	124	77	124	94	124
10	124	27	124	44	124	61	124	78	124	95	124
11	124	28	124	45	124	62	124	79	124	96	124
12	124	29	124	46	124	63	124	80	124	97	124
13	124	30	124	47	124	64	124	81	124	98	124
14	124	31	124	48	124	65	124	82	124	99	124
15	124	32	124	49	124	66	124	83	124	100	124
16	124	33	124	50	124	67	124	84	124		

9. Numerical monoid generated by 241, 251 and 254



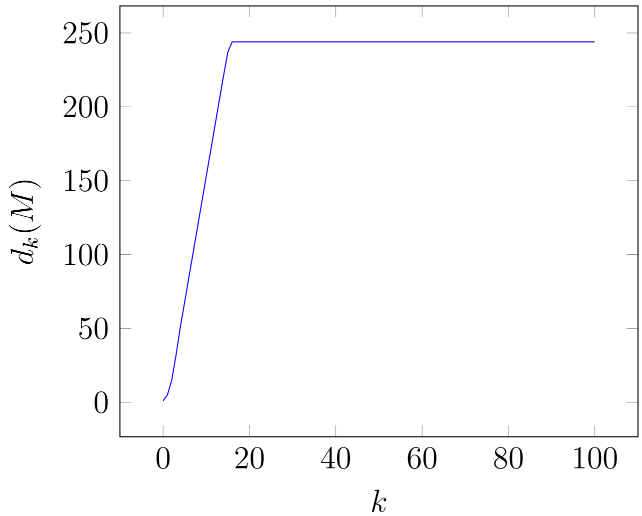
k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$
0	1	17	156	34	241	51	241	68	241	85	241
1	3	18	169	35	241	52	241	69	241	86	241
2	6	19	181	36	241	53	241	70	241	87	241
3	10	20	192	37	241	54	241	71	241	88	241
4	15	21	202	38	241	55	241	72	241	89	241
5	21	22	211	39	241	56	241	73	241	90	241
6	28	23	219	40	241	57	241	74	241	91	241
7	36	24	226	41	241	58	241	75	241	92	241
8	45	25	232	42	241	59	241	76	241	93	241
9	55	26	237	43	241	60	241	77	241	94	241
10	66	27	240	44	241	61	241	78	241	95	241
11	78	28	241	45	241	62	241	79	241	96	241
12	91	29	241	46	241	63	241	80	241	97	241
13	104	30	241	47	241	64	241	81	241	98	241
14	117	31	241	48	241	65	241	82	241	99	241
15	130	32	241	49	241	66	241	83	241	100	241
16	143	33	241	50	241	67	241	84	241		

10. Numerical monoid generated by 243, 247, 257 and 266



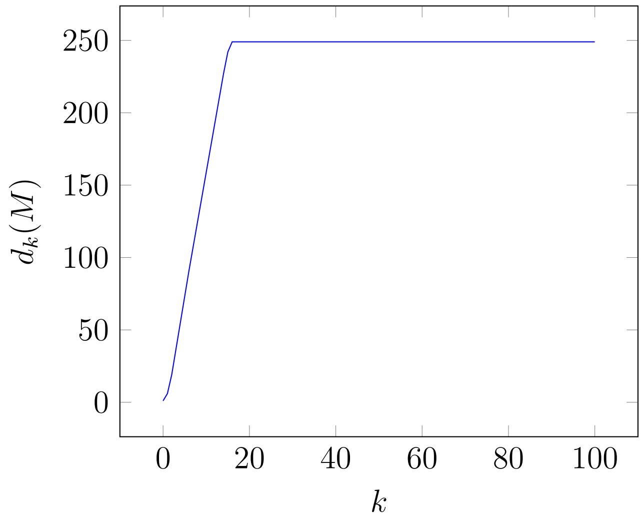
k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$
0	1	17	243	34	243	51	243	68	243	85	243
1	4	18	243	35	243	52	243	69	243	86	243
2	10	19	243	36	243	53	243	70	243	87	243
3	20	20	243	37	243	54	243	71	243	88	243
4	34	21	243	38	243	55	243	72	243	89	243
5	52	22	243	39	243	56	243	73	243	90	243
6	74	23	243	40	243	57	243	74	243	91	243
7	99	24	243	41	243	58	243	75	243	92	243
8	125	25	243	42	243	59	243	76	243	93	243
9	150	26	243	43	243	60	243	77	243	94	243
10	174	27	243	44	243	61	243	78	243	95	243
11	197	28	243	45	243	62	243	79	243	96	243
12	218	29	243	46	243	63	243	80	243	97	243
13	233	30	243	47	243	64	243	81	243	98	243
14	241	31	243	48	243	65	243	82	243	99	243
15	243	32	243	49	243	66	243	83	243	100	243
16	243	33	243	50	243	67	243	84	243		

11. Numerical monoid generated by 244, 248, 253, 255 and 261



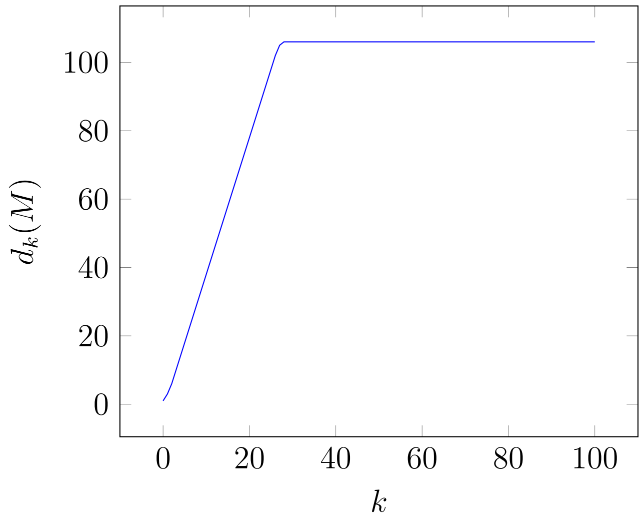
k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$
0	1	17	244	34	244	51	244	68	244	85	244
1	5	18	244	35	244	52	244	69	244	86	244
2	15	19	244	36	244	53	244	70	244	87	244
3	32	20	244	37	244	54	244	71	244	88	244
4	51	21	244	38	244	55	244	72	244	89	244
5	68	22	244	39	244	56	244	73	244	90	244
6	85	23	244	40	244	57	244	74	244	91	244
7	102	24	244	41	244	58	244	75	244	92	244
8	119	25	244	42	244	59	244	76	244	93	244
9	136	26	244	43	244	60	244	77	244	94	244
10	153	27	244	44	244	61	244	78	244	95	244
11	170	28	244	45	244	62	244	79	244	96	244
12	187	29	244	46	244	63	244	80	244	97	244
13	204	30	244	47	244	64	244	81	244	98	244
14	221	31	244	48	244	65	244	82	244	99	244
15	237	32	244	49	244	66	244	83	244	100	244
16	244	33	244	50	244	67	244	84	244		

12. Numerical monoid generated by 249, 255, 257, 259, 265 and 266



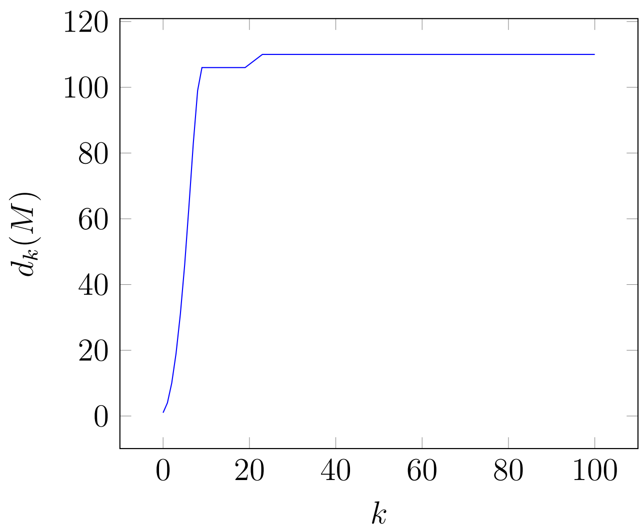
k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$
0	1	17	249	34	249	51	249	68	249	85	249
1	6	18	249	35	249	52	249	69	249	86	249
2	19	19	249	36	249	53	249	70	249	87	249
3	37	20	249	37	249	54	249	71	249	88	249
4	55	21	249	38	249	55	249	72	249	89	249
5	73	22	249	39	249	56	249	73	249	90	249
6	91	23	249	40	249	57	249	74	249	91	249
7	108	24	249	41	249	58	249	75	249	92	249
8	125	25	249	42	249	59	249	76	249	93	249
9	142	26	249	43	249	60	249	77	249	94	249
10	159	27	249	44	249	61	249	78	249	95	249
11	176	28	249	45	249	62	249	79	249	96	249
12	193	29	249	46	249	63	249	80	249	97	249
13	210	30	249	47	249	64	249	81	249	98	249
14	227	31	249	48	249	65	249	82	249	99	249
15	242	32	249	49	249	66	249	83	249	100	249
16	249	33	249	50	249	67	249	84	249		

13. Numerical monoid generated by 106, 113 and 136



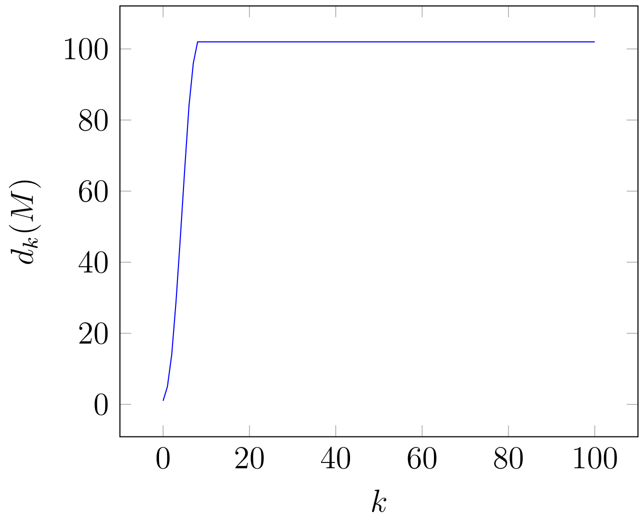
k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$
0	1	17	66	34	106	51	106	68	106	85	106
1	3	18	70	35	106	52	106	69	106	86	106
2	6	19	74	36	106	53	106	70	106	87	106
3	10	20	78	37	106	54	106	71	106	88	106
4	14	21	82	38	106	55	106	72	106	89	106
5	18	22	86	39	106	56	106	73	106	90	106
6	22	23	90	40	106	57	106	74	106	91	106
7	26	24	94	41	106	58	106	75	106	92	106
8	30	25	98	42	106	59	106	76	106	93	106
9	34	26	102	43	106	60	106	77	106	94	106
10	38	27	105	44	106	61	106	78	106	95	106
11	42	28	106	45	106	62	106	79	106	96	106
12	46	29	106	46	106	63	106	80	106	97	106
13	50	30	106	47	106	64	106	81	106	98	106
14	54	31	106	48	106	65	106	82	106	99	106
15	58	32	106	49	106	66	106	83	106	100	106
16	62	33	106	50	106	67	106	84	106		

14. Numerical monoid generated by 110, 111, 134 and 136



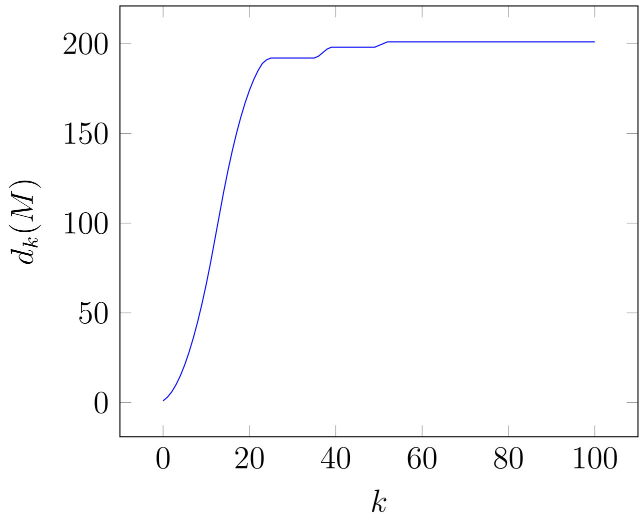
k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$
0	1	17	106	34	110	51	110	68	110	85	110
1	4	18	106	35	110	52	110	69	110	86	110
2	10	19	106	36	110	53	110	70	110	87	110
3	19	20	107	37	110	54	110	71	110	88	110
4	31	21	108	38	110	55	110	72	110	89	110
5	46	22	109	39	110	56	110	73	110	90	110
6	64	23	110	40	110	57	110	74	110	91	110
7	83	24	110	41	110	58	110	75	110	92	110
8	99	25	110	42	110	59	110	76	110	93	110
9	106	26	110	43	110	60	110	77	110	94	110
10	106	27	110	44	110	61	110	78	110	95	110
11	106	28	110	45	110	62	110	79	110	96	110
12	106	29	110	46	110	63	110	80	110	97	110
13	106	30	110	47	110	64	110	81	110	98	110
14	106	31	110	48	110	65	110	82	110	99	110
15	106	32	110	49	110	66	110	83	110	100	110
16	106	33	110	50	110	67	110	84	110		

15. Numerical monoid generated by 102, 117, 121, 123 and 138



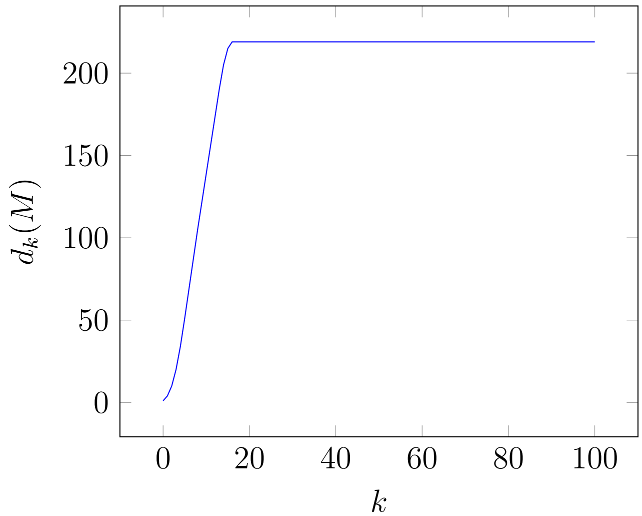
k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$
0	1	17	102	34	102	51	102	68	102	85	102
1	5	18	102	35	102	52	102	69	102	86	102
2	14	19	102	36	102	53	102	70	102	87	102
3	29	20	102	37	102	54	102	71	102	88	102
4	47	21	102	38	102	55	102	72	102	89	102
5	66	22	102	39	102	56	102	73	102	90	102
6	84	23	102	40	102	57	102	74	102	91	102
7	96	24	102	41	102	58	102	75	102	92	102
8	102	25	102	42	102	59	102	76	102	93	102
9	102	26	102	43	102	60	102	77	102	94	102
10	102	27	102	44	102	61	102	78	102	95	102
11	102	28	102	45	102	62	102	79	102	96	102
12	102	29	102	46	102	63	102	80	102	97	102
13	102	30	102	47	102	64	102	81	102	98	102
14	102	31	102	48	102	65	102	82	102	99	102
15	102	32	102	49	102	66	102	83	102	100	102
16	102	33	102	50	102	67	102	84	102		

16. Numerical monoid generated by 201, 212 and 291



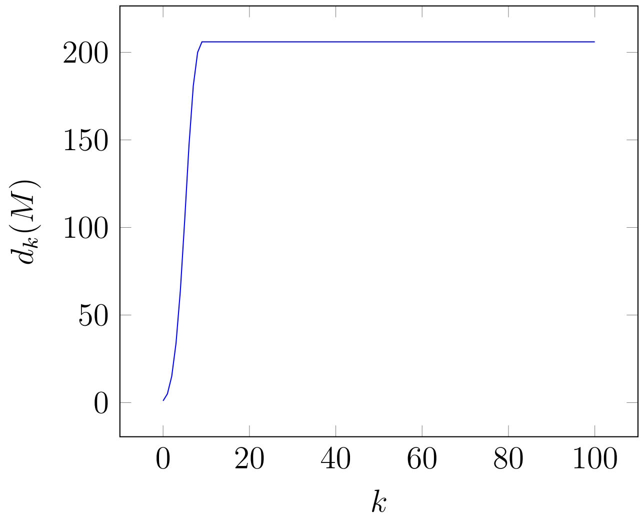
k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$
0	1	17	150	34	192	51	200	68	201	85	201
1	3	18	159	35	192	52	201	69	201	86	201
2	6	19	167	36	193	53	201	70	201	87	201
3	10	20	174	37	195	54	201	71	201	88	201
4	15	21	180	38	197	55	201	72	201	89	201
5	21	22	185	39	198	56	201	73	201	90	201
6	28	23	189	40	198	57	201	74	201	91	201
7	36	24	191	41	198	58	201	75	201	92	201
8	45	25	192	42	198	59	201	76	201	93	201
9	55	26	192	43	198	60	201	77	201	94	201
10	66	27	192	44	198	61	201	78	201	95	201
11	78	28	192	45	198	62	201	79	201	96	201
12	91	29	192	46	198	63	201	80	201	97	201
13	104	30	192	47	198	64	201	81	201	98	201
14	117	31	192	48	198	65	201	82	201	99	201
15	129	32	192	49	198	66	201	83	201	100	201
16	140	33	192	50	199	67	201	84	201		

17. Numerical monoid generated by 219, 231, 267 and 287



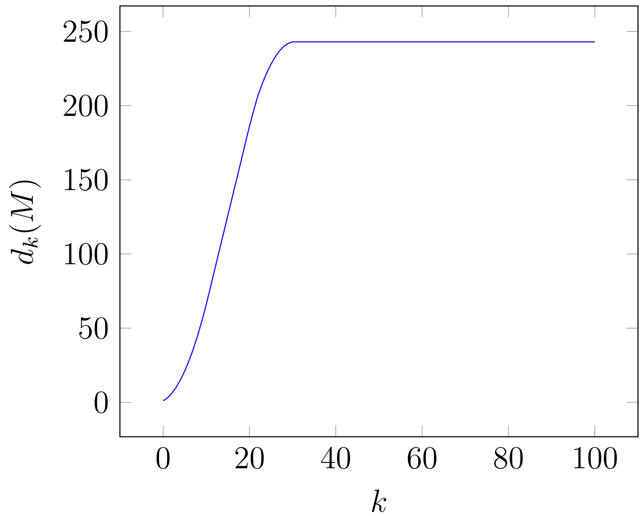
k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$
0	1	17	219	34	219	51	219	68	219	85	219
1	4	18	219	35	219	52	219	69	219	86	219
2	10	19	219	36	219	53	219	70	219	87	219
3	20	20	219	37	219	54	219	71	219	88	219
4	34	21	219	38	219	55	219	72	219	89	219
5	51	22	219	39	219	56	219	73	219	90	219
6	69	23	219	40	219	57	219	74	219	91	219
7	87	24	219	41	219	58	219	75	219	92	219
8	105	25	219	42	219	59	219	76	219	93	219
9	122	26	219	43	219	60	219	77	219	94	219
10	139	27	219	44	219	61	219	78	219	95	219
11	156	28	219	45	219	62	219	79	219	96	219
12	173	29	219	46	219	63	219	80	219	97	219
13	190	30	219	47	219	64	219	81	219	98	219
14	205	31	219	48	219	65	219	82	219	99	219
15	215	32	219	49	219	66	219	83	219	100	219
16	219	33	219	50	219	67	219	84	219		

18. Numerical monoid generated by 206, 214, 238, 247 and 265



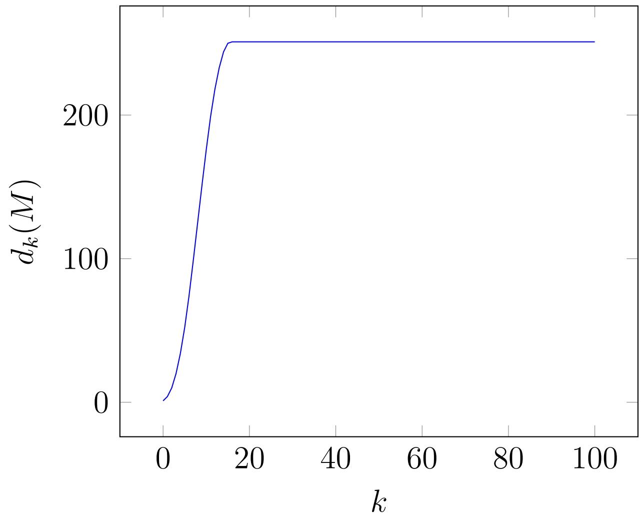
k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$
0	1	17	206	34	206	51	206	68	206	85	206
1	5	18	206	35	206	52	206	69	206	86	206
2	15	19	206	36	206	53	206	70	206	87	206
3	34	20	206	37	206	54	206	71	206	88	206
4	64	21	206	38	206	55	206	72	206	89	206
5	104	22	206	39	206	56	206	73	206	90	206
6	147	23	206	40	206	57	206	74	206	91	206
7	181	24	206	41	206	58	206	75	206	92	206
8	200	25	206	42	206	59	206	76	206	93	206
9	206	26	206	43	206	60	206	77	206	94	206
10	206	27	206	44	206	61	206	78	206	95	206
11	206	28	206	45	206	62	206	79	206	96	206
12	206	29	206	46	206	63	206	80	206	97	206
13	206	30	206	47	206	64	206	81	206	98	206
14	206	31	206	48	206	65	206	82	206	99	206
15	206	32	206	49	206	66	206	83	206	100	206
16	206	33	206	50	206	67	206	84	206		

19. Numerical monoid generated by 243, 245 and 267



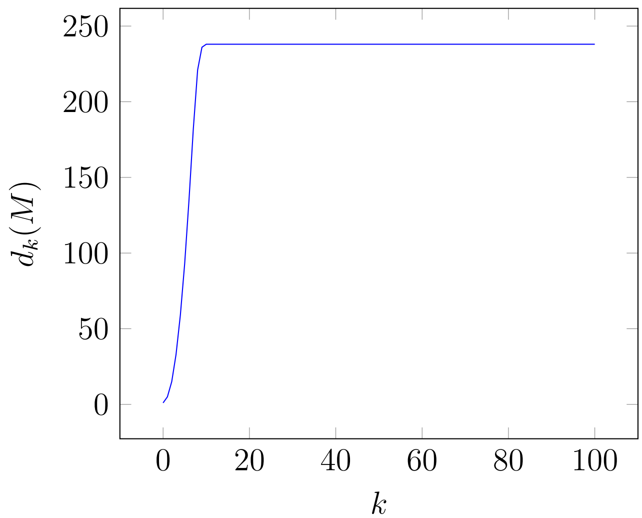
k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$
0	1	17	150	34	243	51	243	68	243	85	243
1	3	18	162	35	243	52	243	69	243	86	243
2	6	19	174	36	243	53	243	70	243	87	243
3	10	20	186	37	243	54	243	71	243	88	243
4	15	21	197	38	243	55	243	72	243	89	243
5	21	22	207	39	243	56	243	73	243	90	243
6	28	23	215	40	243	57	243	74	243	91	243
7	36	24	222	41	243	58	243	75	243	92	243
8	45	25	228	42	243	59	243	76	243	93	243
9	55	26	233	43	243	60	243	77	243	94	243
10	66	27	237	44	243	61	243	78	243	95	243
11	78	28	240	45	243	62	243	79	243	96	243
12	90	29	242	46	243	63	243	80	243	97	243
13	102	30	243	47	243	64	243	81	243	98	243
14	114	31	243	48	243	65	243	82	243	99	243
15	126	32	243	49	243	66	243	83	243	100	243
16	138	33	243	50	243	67	243	84	243		

20. Numerical monoid generated by 251, 268, 272 and 277



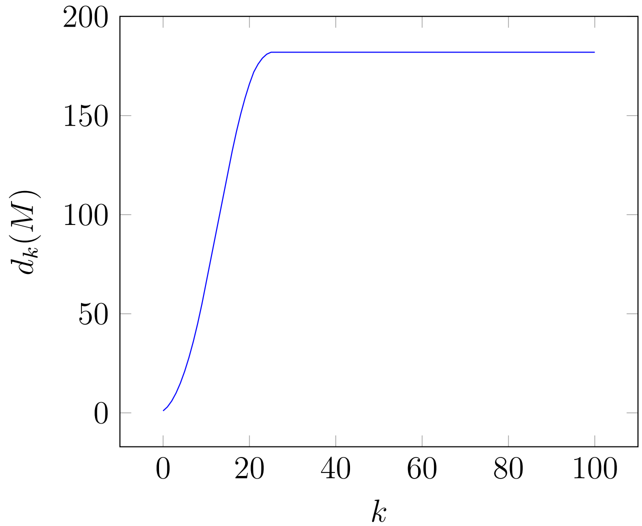
k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$
0	1	17	251	34	251	51	251	68	251	85	251
1	4	18	251	35	251	52	251	69	251	86	251
2	10	19	251	36	251	53	251	70	251	87	251
3	20	20	251	37	251	54	251	71	251	88	251
4	34	21	251	38	251	55	251	72	251	89	251
5	52	22	251	39	251	56	251	73	251	90	251
6	74	23	251	40	251	57	251	74	251	91	251
7	99	24	251	41	251	58	251	75	251	92	251
8	125	25	251	42	251	59	251	76	251	93	251
9	151	26	251	43	251	60	251	77	251	94	251
10	176	27	251	44	251	61	251	78	251	95	251
11	199	28	251	45	251	62	251	79	251	96	251
12	218	29	251	46	251	63	251	80	251	97	251
13	233	30	251	47	251	64	251	81	251	98	251
14	244	31	251	48	251	65	251	82	251	99	251
15	250	32	251	49	251	66	251	83	251	100	251
16	251	33	251	50	251	67	251	84	251		

21. Numerical monoid generated by 238, 241, 247, 266 and 279



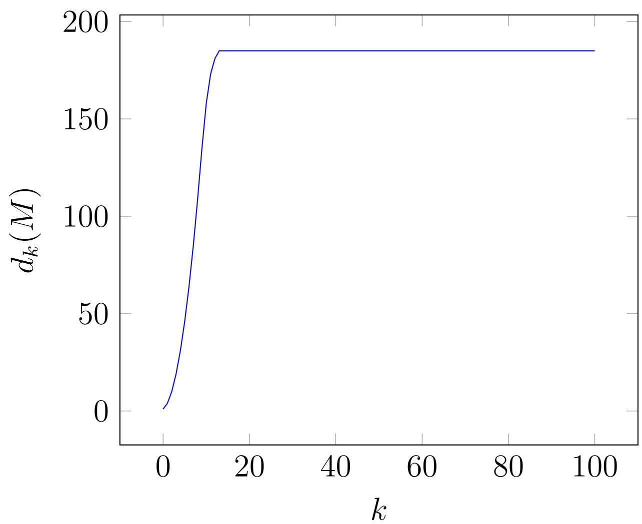
k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$
0	1	17	238	34	238	51	238	68	238	85	238
1	5	18	238	35	238	52	238	69	238	86	238
2	15	19	238	36	238	53	238	70	238	87	238
3	33	20	238	37	238	54	238	71	238	88	238
4	59	21	238	38	238	55	238	72	238	89	238
5	93	22	238	39	238	56	238	73	238	90	238
6	135	23	238	40	238	57	238	74	238	91	238
7	182	24	238	41	238	58	238	75	238	92	238
8	221	25	238	42	238	59	238	76	238	93	238
9	236	26	238	43	238	60	238	77	238	94	238
10	238	27	238	44	238	61	238	78	238	95	238
11	238	28	238	45	238	62	238	79	238	96	238
12	238	29	238	46	238	63	238	80	238	97	238
13	238	30	238	47	238	64	238	81	238	98	238
14	238	31	238	48	238	65	238	82	238	99	238
15	238	32	238	49	238	66	238	83	238	100	238
16	238	33	238	50	238	67	238	84	238		

22. Numerical monoid generated by 182, 187 and 193



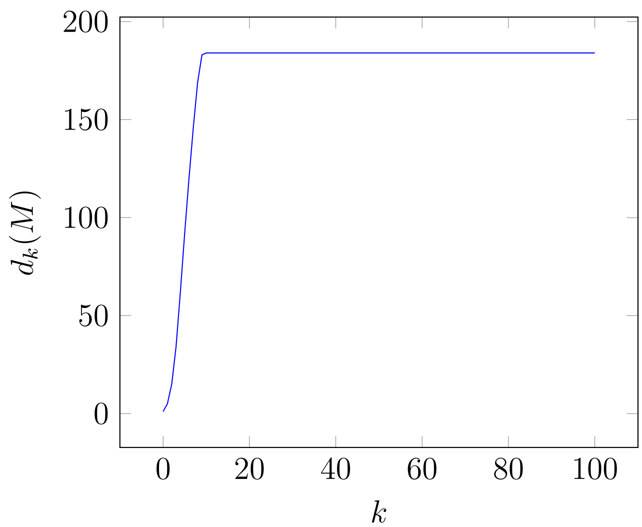
k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$
0	1	17	142	34	182	51	182	68	182	85	182
1	3	18	151	35	182	52	182	69	182	86	182
2	6	19	159	36	182	53	182	70	182	87	182
3	10	20	166	37	182	54	182	71	182	88	182
4	15	21	172	38	182	55	182	72	182	89	182
5	21	22	176	39	182	56	182	73	182	90	182
6	28	23	179	40	182	57	182	74	182	91	182
7	36	24	181	41	182	58	182	75	182	92	182
8	45	25	182	42	182	59	182	76	182	93	182
9	55	26	182	43	182	60	182	77	182	94	182
10	66	27	182	44	182	61	182	78	182	95	182
11	77	28	182	45	182	62	182	79	182	96	182
12	88	29	182	46	182	63	182	80	182	97	182
13	99	30	182	47	182	64	182	81	182	98	182
14	110	31	182	48	182	65	182	82	182	99	182
15	121	32	182	49	182	66	182	83	182	100	182
16	132	33	182	50	182	67	182	84	182		

23. Numerical monoid generated by 185, 198, 207 and 209



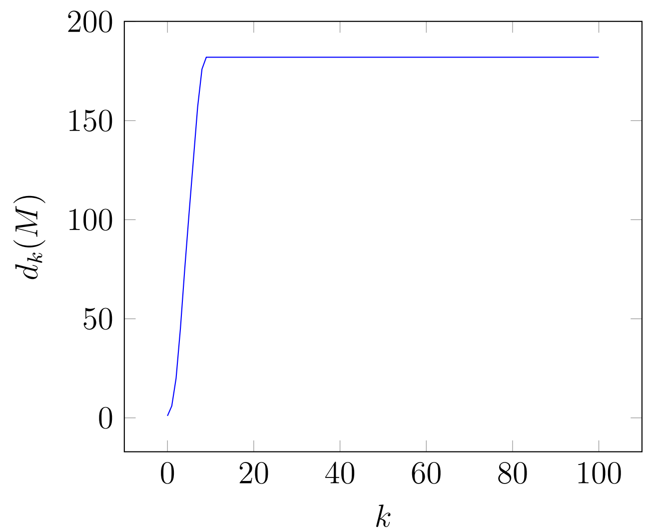
k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$
0	1	17	185	34	185	51	185	68	185	85	185
1	4	18	185	35	185	52	185	69	185	86	185
2	10	19	185	36	185	53	185	70	185	87	185
3	19	20	185	37	185	54	185	71	185	88	185
4	31	21	185	38	185	55	185	72	185	89	185
5	46	22	185	39	185	56	185	73	185	90	185
6	64	23	185	40	185	57	185	74	185	91	185
7	85	24	185	41	185	58	185	75	185	92	185
8	109	25	185	42	185	59	185	76	185	93	185
9	135	26	185	43	185	60	185	77	185	94	185
10	158	27	185	44	185	61	185	78	185	95	185
11	173	28	185	45	185	62	185	79	185	96	185
12	181	29	185	46	185	63	185	80	185	97	185
13	185	30	185	47	185	64	185	81	185	98	185
14	185	31	185	48	185	65	185	82	185	99	185
15	185	32	185	49	185	66	185	83	185	100	185
16	185	33	185	50	185	67	185	84	185		

24. Numerical monoid generated by 184, 191, 195, 207 and 208



k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$
0	1	17	184	34	184	51	184	68	184	85	184
1	5	18	184	35	184	52	184	69	184	86	184
2	15	19	184	36	184	53	184	70	184	87	184
3	34	20	184	37	184	54	184	71	184	88	184
4	62	21	184	38	184	55	184	72	184	89	184
5	92	22	184	39	184	56	184	73	184	90	184
6	120	23	184	40	184	57	184	74	184	91	184
7	146	24	184	41	184	58	184	75	184	92	184
8	169	25	184	42	184	59	184	76	184	93	184
9	183	26	184	43	184	60	184	77	184	94	184
10	184	27	184	44	184	61	184	78	184	95	184
11	184	28	184	45	184	62	184	79	184	96	184
12	184	29	184	46	184	63	184	80	184	97	184
13	184	30	184	47	184	64	184	81	184	98	184
14	184	31	184	48	184	65	184	82	184	99	184
15	184	32	184	49	184	66	184	83	184	100	184
16	184	33	184	50	184	67	184	84	184		

25. Numerical monoid generated by 182, 190, 199, 201, 206 and 209



k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$	k	$d_k(M)$
0	1	17	182	34	182	51	182	68	182	85	182
1	6	18	182	35	182	52	182	69	182	86	182
2	20	19	182	36	182	53	182	70	182	87	182
3	45	20	182	37	182	54	182	71	182	88	182
4	75	21	182	38	182	55	182	72	182	89	182
5	103	22	182	39	182	56	182	73	182	90	182
6	130	23	182	40	182	57	182	74	182	91	182
7	157	24	182	41	182	58	182	75	182	92	182
8	176	25	182	42	182	59	182	76	182	93	182
9	182	26	182	43	182	60	182	77	182	94	182
10	182	27	182	44	182	61	182	78	182	95	182
11	182	28	182	45	182	62	182	79	182	96	182
12	182	29	182	46	182	63	182	80	182	97	182
13	182	30	182	47	182	64	182	81	182	98	182
14	182	31	182	48	182	65	182	82	182	99	182
15	182	32	182	49	182	66	182	83	182	100	182
16	182	33	182	50	182	67	182	84	182		

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.

[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.jpllnet.sfsu.edu/10.1007/978-1-4612-5350-1.

[Ram05] J. L. Ramírez Alfonsın. *The Diophantine Frobenius problem*. Vol. 30. Oxford Lecture Series in Mathematics and its Applications. Oxford University Press, Oxford, 2005, pp. xvi+243. ISBN: 978-0-19-856820-9; 0-19-856820-7. DOI: 10.1093/acprof:oso/9780198568209.001.0001. URL: https://doi-org.jpllnet.sfsu.edu/10.1093/acprof:oso/9780198568209.001.0001.

[BG09] Winfried Bruns and Joseph Gubeladze. *Polytopes, rings, and K-theory*. Springer Monographs in Mathematics. New York: Springer-Verlag, 2009.

[RG09] J. C. Rosales and P. A. García-Sánchez. *Numerical semigroups*. Vol. 20. Developments in Mathematics. Springer, New York, 2009, pp. x+181. ISBN: 978-1-4419-0159-0. DOI: 10.1007/978-1-4419-0160-6. URL: https://doi-org.jpllnet.sfsu.edu/10.1007/978-1-4419-0160-6.

[Isk11] Aicke Hinrichs Iskander Aliev Martin Henk. "Expected Frobenius number". In: *Journal of Combinatorial Theory, Series A* 118.12 (2011), pp. 525–531. DOI: 10.1016/j.jcta.2009.12.012. URL: https://doi.org/10.1016/j.jcta.2009.12.012.

[ONe17] Christopher O'Neill. "On factorization invariants and Hilbert functions". In: *J. Pure Appl. Algebra* 221.12 (2017), pp. 3069–3088. ISSN: 0022-4049. DOI: 10.1016/j.jpaa.2017.02.014. URL: https://doi-org.jpllnet.sfsu.edu/10.1016/j.jpaa.2017.02.014.

[Gub] Joseph Gubeladze. *Bottom complexes*. Preprint (2022).

[ADG20] Abdallah Assi, Marco D'Anna, and Pedro A. García-Sánchez. *Numerical semigroups and applications*. Vol. 3. RSME Springer Series. Second edition [of 3558713]. Springer, Cham, [2020] ©2020, pp. xiv+138. ISBN: 978-3-030-54942-8; 978-3-030-54943-5. DOI: `10.1007/978-3-030-54943-5`. URL: `https://doi-org.jpllnet.sfsu.edu/10.1007/978-3-030-54943-5`.

[CGO20] Scott Chapman, Rebecca Garcia, and Christopher O'Neill. "Beyond coins, stamps, and Chicken McNuggets: an invitation to numerical semigroups". In: *A project-based guide to undergraduate research in mathematics—starting and sustaining accessible undergraduate research*. Found. Undergrad. Res. Math. Birkhäuser/Springer, Cham, [2020] ©2020, pp. 177–202. DOI: `10.1007/978-3-030-37853-0\_6`. URL: `https://doi-org.jpllnet.sfsu.edu/10.1007/978-3-030-37853-0_6`.