

Algebraic Geometry Curves and Elkies Codes

Bárbara Rivera

University of Puerto Rico - Río Piedras

Eric Summerville

St. Mary's University

Jessica Zúñiga

Rice University

July 27, 2001

Abstract

Error control codes are widely used to increase the reliability of transmission of information over various forms of communications channels. *Algebraic geometry (AG)* codes were introduced by V. Goppa in the 1980's. The codewords in an AG code are obtained by evaluating functions at the points of an algebraic curve over a finite field; Goppa's construction produces some of the best codes currently known. N. Elkies has recently proposed a generalization of this construction that produces non-linear codes with even better parameters. Their non-linearity is a serious drawback for their use in applications. Our research involves analyzing Elkies codes on the line and on Hermitian curves over finite fields. We show that Elkies codes contain relatively large linear subcodes, and examine the codeword recognition problem using properties of these linear subcodes.

1 Introduction

In the past twenty years there have been many advances in the field of coding theory. One development concerns the work of three scientists: Tsfasman, Vladut, and Zink, who constructed long codes over finite fields that exceeded the Gilbert-Varshamov bound. Their discovery was based on the work of the Russian mathematician V.D. Goppa who proposed the idea of constructing codes by evaluating collections of rational functions with specified poles on algebraic curves over finite fields in order to get better codes. The construction can be seen as a generalization of the idea of evaluating polynomials at certain points on a straight line as it is done in the construction of the Reed-Solomon codes [4]. Very recently, a Harvard University professor by the name of Noam D. Elkies proposed a construction of non-linear codes from algebraic curves that sometimes yield superior results. "Elkies codes", as we will call them, are constructed by evaluating collections of rational functions that are more general than those used in the Goppa construction. The functions used to produce an Elkies code have specified degree bounds on their divisors of zeroes and poles. Due to their

non-linearity, several obstacles arise when dealing with the application of these codes. One such obstacle is simply the problem of recognizing when a word is one of the codewords of an Elkie code. Elkie codes do not have a parity-check matrix that can be used to detect when a word is a codeword. Another obstacle is trying to find an efficient way to encode/decode Elkie codes. The focus of our research deals with understanding Goppa codes and analyzing Elkie codes.

2 Review of Reed-Solomon Codes

Definition 1 Label the $q - 1$ nonzero elements of $GF(q)$ as $\{\beta_1, \dots, \beta_{q-1}\}$. Also let $S = \text{Span}\{1, t, \dots, t^{k-1}\}$ be the vector space of all polynomials of degree $< k$ with coefficients in $GF(q)$. The **Reed-Solomon code** $RS(k, q)$ may be obtained by evaluating the polynomials $f(x) \in S$ at the points of $GF(q)/\{0\}$. That is, the Reed-Solomon Code $RS(k, q)$ is defined to be

$$RS(k, q) := \{(f(\beta_1), \dots, f(\beta_{q-1})) \mid f \in S\}.$$

For future reference, we note that it would also be possible to define a code as follows. Let $\omega_1, \dots, \omega_{q-1}$ be any nonzero elements of $GF(q)$. Let

$$GRS(k, q) := \{(\omega_1 f(\beta_1), \dots, \omega_{q-1} f(\beta_{q-1})) \mid f \in S\}.$$

The resulting code is called a **generalized Reed-Solomon code**.

3 A First Generalization of Reed-Solomon Codes: Hermitian Goppa Codes

In this section we will consider a concrete example of the Goppa codes mentioned in the introduction. We start by introducing a particular family of algebraic curves with interesting properties.

Definition 2 Let m be a power of some prime number. The **Hermitian curve** over $GF(m^2)$ is the curve with the defining equation

$$x^{m+1} = y^m + y.$$

This curve has $N = m^3$ points on the affine plane over $GF(m^2)$, denoted Q_1, \dots, Q_N , plus one point at infinity, denoted by Q_∞ . In fact the Hermitian curve has the maximum possible number of points with coordinates in $GF(m^2)$ for a curve of degree $m + 1$ defined over $GF(m^2)$.

We will next show how to define a particular type of Goppa code on Hermitian curves.

Definition 3 Let H_{m^2} be the set of points of the Hermitian curve over $GF(m^2)$. Given any integer $a \geq 0$, define the set

$$L(aQ_\infty) = \text{Span}\{x^r y^s \mid rm + s(m+1) \leq a\}.$$

We get a Goppa code over the Hermitian curve by evaluating all functions $f \in L(aQ_\infty)$ at the points in $\{Q_1, \dots, Q_N\}$. The **Hermitian Goppa code** $C(a)$ is the image of the evaluation map

$$\begin{aligned} ev : L(aQ_\infty) &\rightarrow GF(m^2)^N \\ f &\mapsto (f(Q_1), \dots, f(Q_N)) \end{aligned}$$

The construction of the Goppa codes can be seen as a generalization of the Reed-Solomon codes which evaluate functions at points on a straight line. Goppa proposes that one should instead evaluate these functions on algebraic curves such as the Hermitian curve in order to get longer codes with better parameters.

In some of the following steps of the next example we will use the following theorem from algebraic geometry.

Theorem 1 (*Bezout's Theorem*) If $f, g \in k[x, y]$ are polynomials of total degree d and e respectively, then the curves $C_f = \{(x, y) \in k^2 : f(x, y) = 0\}$ and $C_g = \{(x, y) \in k^2 : g(x, y) = 0\}$ intersect in at most de points, provided that f and g have no common nonconstant factors. Furthermore, if k is algebraically closed, the projective closures of C_f and C_g , \widehat{C}_f and \widehat{C}_g , intersect in exactly $d * e$ points of $\mathbb{P}^2(k)$ when points are counted with multiplicity.

Example. We construct Hermitian Goppa codes using the curve over the finite field $GF(4)$ (that is, in the case $m = 2$). Changing the parameter “a” allows us to find the minimum distance and the dimension of the code.

Let β be a primitive element of $GF(4)$. Then every non-zero element in $GF(4)$ can be represented by some power of β , such that

$$0 = 0, \beta^0 = 1, \beta = \beta, \beta^2 = \beta + 1$$

The corresponding Hermitian curve equation for this case is $x^3 = y^2 + y$. We need to find the set of points with coordinates in $GF(4)$ that satisfy this equation. We will call this set

$$H_4 = \{(x, y) \in (GF(4))^2 \mid x^3 = y^2 + y\}$$

By substituting each of the four different possible values for x and solving the resulting equations for y , we obtain:

$$H_4 = \{(0, 0), (0, 1), (1, \beta), (1, \beta^2), (\beta, \beta), (\beta, \beta^2), (\beta^2, \beta), (\beta^2, \beta^2)\}$$

(and also one additional point at infinity).

We will use the following graph of H_4 to calculate the minimum distance of the Hermitian Goppa codes.

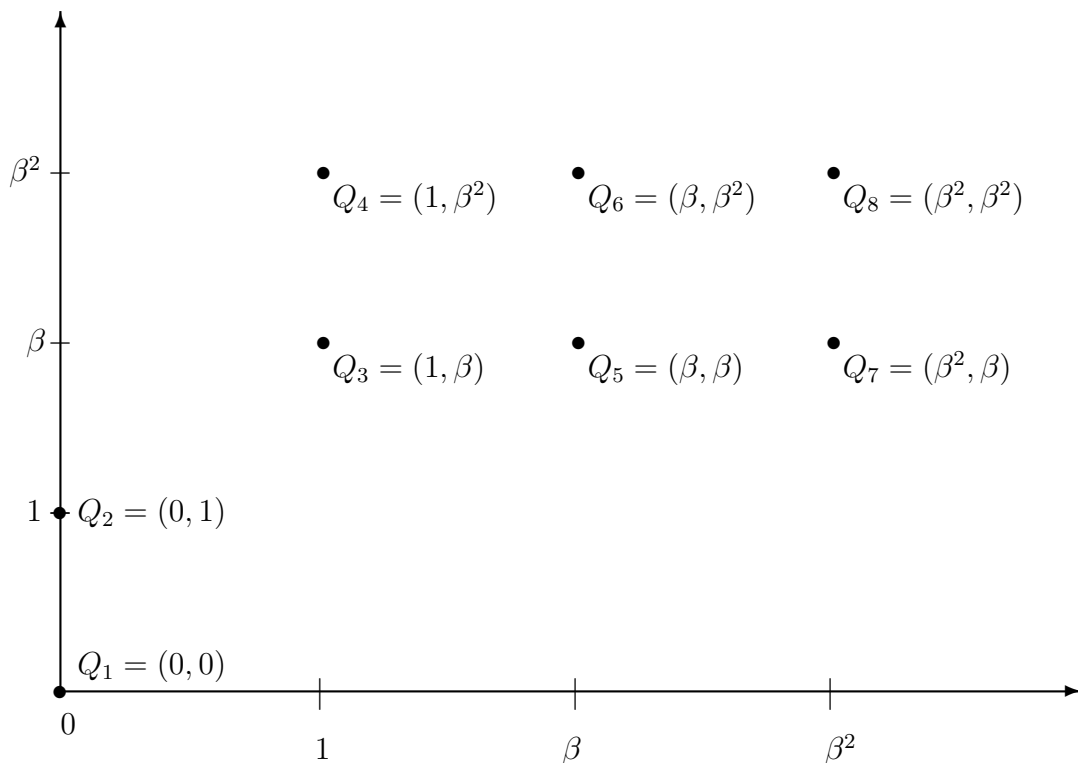


Figure 1. Solutions of $x^3 = y^2 + y$ in $GF(4)^2$

Now we will study the Hermitian Goppa codes $C(a)$ for different values of a , starting at one and getting no larger than the length of our words, n . The length is the total number of components in our word. This is, the number of points in the curve. For $GF(4)$, $n = 8$. Using the definition $L(aQ_\infty) = \text{Span}\{x^r y^s \mid 2r + 3s \leq a\}$, the functions that generate the space can be calculated so that the solution points can be evaluated to obtain our codewords.

Let $a = 1$. The set of functions obtained is $\{1\}$. The dimension of the code is $k = 1$. The only non-zero codewords are $\{11111111\}$ and its multiples, so $d = 8$.

Let $a = 2$. The set of functions obtained is $\{1, x\}$. The dimension of the code is $k = 2$. Evaluating $\alpha_1 + \alpha_2 x$ gives codewords with zeroes at points with $x = \frac{\alpha_1}{\alpha_2}$. There are two such points for all α_1, α_2 , so the minimum distance is $8 - 2 = 6$.

Let $a = 3$. The set of functions obtained is $\{1, x, y\}$. The dimension of the code is $k = 3$. Evaluating $\alpha_1 + \alpha_2 x + \alpha_3 y$ gives codewords with zeroes at points with $y = \frac{\alpha_2}{\alpha_3} x + \frac{\alpha_1}{\alpha_3}$. By direct computation, or by applying Bezout's theorem, there are at

most three such points for each f of this form. Moreover, some f 's, such as $f = y + \beta$ are zero at three different points, so the minimum distance is $8 - 3 = 5$.

Let $a = 4$. The set of functions obtained is $\{1, x, y, x^2\}$. The dimension of the code is $k = 4$. Evaluating $f = \alpha_1 + \alpha_2x + \alpha_3y + \alpha_4x^2$ gives codewords with zeroes at most at four points. Since we can find four such points then the minimum distance is $8 - 4 = 4$. (Note: in this case Bézout's theorem gives the upper bound $3 \cdot 2 = 6$ for the number of common solutions of the Hermitian curve equation and $f = 0$, but the bound is not achieved because of the special form of f .)

Let $a = 5$. The set of functions obtained is $\{1, x, y, x^2, xy\}$. The dimension of the code is $k = 5$. Evaluating $f = \alpha_1 + \alpha_2x + \alpha_3y + \alpha_4x^2 + \alpha_5xy$ gives codewords with zeroes at most six points, by Bézout's theorem. Since we can find only five such points the minimum distance is $8 - 5 = 3$.

Let $a = 6$. The set of functions obtained is $\{1, x, y, x^2, xy, y^2\}$. The dimension of the code is $k = 6$, Evaluating $\alpha_1 + \alpha_2x + \alpha_3y + \alpha_4x^2 + \alpha_5xy + \alpha_6y^2$ gives codewords with zeroes at most at six points. Since we can find six such points the minimum distance is $8 - 6 = 2$.

Let $a = 7$. The set of functions obtained is $\{1, x, y, x^2, xy, y^2, x^2y, x^3\}$. However, these monomials are not linearly independent at the points on the Hermitian curve because $x^3 = y^2 + y$ at these points. The dimension of this code is only $k = 7$. Evaluating

$$f = \alpha_1 + \alpha_2x + \alpha_3y + \alpha_4x^2 + \alpha_5xy + \alpha_6y^2 + \alpha_7x^2y$$

gives codewords with zeroes at most at nine points by Bezout's theorem. But the actual minimum distance is $d = 8 - 7 = 1$.

a	n=length of word	k=dimension	d=distance	Number of Fuctions
1	8	1	8	1
2	8	2	6	2
3	8	3	5	3
4	8	4	4	4
5	8	5	3	5
6	8	6	2	6
7	8	7	1	7

Table 1. Hermitian Goppa codes with $m = 2$.

4 Elkies Codes on a Straight Line

In 2001 Elkies provided a further generalization of the Goppa codes by evaluating rational functions over curves in $GF(q)$ where the numerator and the denominator have a degree bound h .

In the Reed-Solomon codes that we previously studied, we evaluated polynomials only at the points in $GF(q)/\{0\}$. In the Hermitian Goppa codes the monomials $x^r y^s$ are defined at all the points Q_1, \dots, Q_N . They do not give finite values at Q_∞ , the solution point of the Hermitian curve at infinity. A new possibility for the Elkie codes is that we will need to evaluate a function at a point where it does not have a well-defined finite value.

Definition 4 A **pole** for a function $f(x) = \frac{j(x)}{b(x)}$ on the line is any point x_0 where $b(x_0) = 0$ but $j(x_0) \neq 0$, or more generally, a point where the denominator vanishes to a higher degree than the numerator does.

For future reference we will also define a zero of a function.

Definition 5 A **zero** for a function $f(x) = \frac{j(x)}{b(x)}$ is any point x_0 where $j(x_0) = 0$ but $b(x_0) \neq 0$ or more generally, a point where the numerator vanishes to a higher degree than the denominator does.

In Elkie codes on the line it is possible for a codeword to have one or more entries ∞ in positions corresponding to points where the function being evaluated has a pole. The function generating the codeword is also evaluated at ∞ to give one of the entries. So the ‘‘alphabet’’ (the set of possible entries in codewords) and the set of entries in codewords are in one-to-one correspondence with the set we will denote by \mathbb{P}^1 .

We will denote the set of points on the projective line over $GF(q)$ by

$$\mathbb{P}^1 = \{0, 1, \beta, \dots, \beta^{q-2}, \infty\}$$

Evaluating a function at ∞ is equivalent to evaluating it at the point $x = \frac{1}{t}$ where $t = 0$. Even though in a Goppa code over a straight line, i.e. the Reed-Solomon code, the denominator is the constant 1, when we evaluate it at $\frac{1}{t}$ we will multiply the denominator by 0. Consequently, the only possible pole is the point at ∞ . Elkie’s idea to evaluate rational functions over curves allows for poles at points other than ∞ .

We will begin by analyzing the construction of Elkie codes over a straight line.

Definition 6 The **Elkie code** on \mathbb{P}^1 with degree bound h is the set

$$E_h(\mathbb{P}^1) = \{(f(0), f(1), f(\beta), \dots, f(\beta^{q-2}), f(\infty))\}$$

where f runs over the set of all rational functions $f(x) = \frac{p(x)}{q(x)}$ with $\deg(p), \deg(q) \leq h$.

Definition 7 The **divisor** of a function f whose zeroes are z_i with multiplicity m_i , and whose poles are p_j with multiplicity n_j is defined to be

$$\text{div}(f) = m_1 z_1 + \dots + m_r z_r - n_1 p_1 - \dots - n_s p_s, \quad m_i, n_j > 0$$

On any projective curve, for any f , the sum of the multiplicities of the zeroes equals the sum of the multiplicities of the poles.

Example. Let $f(x) = \frac{x^2+1}{x+\beta}$ on \mathbb{P}^1 over the field $GF(4)$. The zeroes of f are points where $x^2 + 1 = 0$ or $x^2 = 1$, so we have one zero at $x = 1$ with a multiplicity of two. Similarly we can find that f has poles at $x = \beta, \infty$. Therefore the divisor of f is $\text{div}(f) = 2(1) - \beta - \infty$.

Note that a rational function as in the definition of the Elkies code $E_h(\mathbb{P}^1)$ will have at most h zeroes and at most h poles. This gives an equivalent form of the definition:

$$E_h(\mathbb{P}^1) = \{(f(0), f(1), f(\beta), \dots, f(\infty))\}$$

where f runs over the set of all rational functions f on \mathbb{P}^1 whose divisor of zeroes and poles has the form

$$\text{div}(f) = D - D'$$

where D and D' have only positive coefficients, and $\text{deg}(D), \text{deg}(D') \leq h$. It is this form that will be used to generalize the definition of the Elkies codes to the Hermitian curve (and other curves).

A fundamental drawback of the Elkies codes is that they are not linear since closure of addition may not hold for codewords generated from two functions with different denominators.

Example. Let $E_1(\mathbb{P}^1)$ be an Elkies code over $GF(4)$. Then $f_1 = \frac{x}{x+1}$ and $f_2 = \frac{1}{x+\beta}$ are two functions that satisfy the degree bounds with $h = 1$, but their sum $\frac{x}{x+1} + \frac{1}{x+\beta} = \frac{x^2+\beta^2x+1}{x^2+\beta^2x+\beta}$ does not. The Elkies codewords obtained by evaluating f_1 and f_2 would not sum to a codeword of the $E_1(\mathbb{P}^1)$ code.

Even though $E_h(\mathbb{P}^1)$ is not linear, due to its construction we are able to find linear subcodes.

Theorem 2 *Consider the code $E_h(\mathbb{P}^1)$ over the field $GF(q)$ and the subcode consisting of words obtained by evaluating functions with a fixed denominator. If this subcode is punctured at the positions corresponding to poles of the functions, the resulting code is a (possibly punctured) extended generalized Reed-Solomon code. There are $\sum_{i=0}^h q^i$ of these codes associated to the Elkies code.*

Proof. For simplicity, we will call the Elkies code with degree bound h , E_h . All words in E_h are created by evaluating functions $f(x) = \frac{p(x)}{q(x)}$ where $p(x)$ and $q(x)$ are of the form $\alpha_h x^h + \dots + \alpha_1 x + \alpha_0, \alpha_i \in GF(q)$.

For any f we can factor out the leading coefficients in the numerator and the denominator to get a function in the form

$$\alpha \left(\frac{x^h + \gamma_1 x^{h-1} + \dots + \gamma_{h-1} x + \gamma_h}{x^d + \beta_1 x^{d-1} + \dots + \beta_d} \right)$$

where α is a constant.

Consider the subsets of E_h consisting of words generated by functions with the same denominator. We will show that linearity properties do hold for this subset. Let S be the subset of functions with denominator $d(x)$. Then for $f_1(x) = \frac{s(x)}{d(x)}$ and $f_2(x) = \frac{h(x)}{d(x)}$, $f_1(x) + f_2(x) = \frac{s(x)}{d(x)} + \frac{f(x)}{d(x)} = \frac{s(x)+f(x)}{d(x)}$. Since s and f have degrees bounded by h , so does their sum. Hence $f_1 + f_2$ will also evaluate to give a codeword in E_h . If we delete entries containing ∞ (in locations where d is zero), then the resulting word is the sum of the words obtained by puncturing the codewords from f_1 and f_2 in the same locations.

Thus codewords generated by functions in the same subset (i.e. functions with similar denominators), after puncturing, can be grouped into linear codes associated to $E_h(\mathbb{P}^1)$. Each of these codes can be generated by evaluating $l(x) \in \text{Span}\{\frac{x^i}{d(x)}\}$ for $i = 0, \dots, h$ at the points where d is not zero.

This reminds us of the generators for the Reed-Solomon code except for the extra factor of $\frac{1}{d(x)}$. Let Q_1, \dots, Q_m be the points of \mathbb{P}^1 that are not zeroes of d . If we let $\omega_i = \frac{1}{d(Q_i)}$, then our words will look like

$$(\omega_1 n(Q_1), \omega_2 n(Q_2), \dots, \omega_m n(Q_m))$$

where $n \in \text{Span}\{1, x, x^2, \dots, x^h\}$ is the numerator of l . The factors of ω_i and the evaluation at infinity, plus the fact that we may have to omit several points where our functions have poles make this code a possibly punctured, extended, generalized Reed-Solomon code. Therefore every possible denominator with a leading coefficient of one will give one of these codes.

The number of possible denominators can be evaluated by a simple counting method. Any polynomial of degree bound h can be expressed as $\alpha_0 x^h + \alpha_1 x^{h-1} + \dots + \alpha_{h-1} x + \alpha_h$. Since we only want to count denominators with one as the leading coefficient, then we take the sum of all possible polynomials with leading terms $\{x^h, x^{h-1}, h^{h-2}, \dots, x, 1\}$. For $x^0 = 1$ there is only one possible polynomial with a coefficient of one, namely 1. For x , there are q possible polynomials: $\{x + 0, x + \beta^0, x + \beta^1, \dots, x + \beta^{q-1}\}$ for β a primitive element for $GF(q)$. For x^2 there are q^2 possibilities since for a polynomial of the form $x^2 + \alpha_1 x + \alpha_2$, $\alpha_1, \alpha_2 \in GF(q)$ there are q choices for each α_1 and α_2 . In general, for x^n there are q^n possible polynomials since for $x^n + \alpha_{n-1} x^{n-1} + \dots + \alpha_1 x + \alpha_0$, there are q choices for each of the n coefficients.

Hence there are $q^n + q^{n-1} + \dots + q^2 + q + 1$ possible denominators of degree bound h . We can say that the total number of denominators for an Elkies code of degree bound h is $p = \sum_{i=0}^h q^i$. Since every distinct polynomial is associated with a unique extended generalized Reed-Solomon code as above, there are p Reed-Solomon codes associated to an Elkies code of degree bound h .

5 Elkies Codes on the Hermitian Curve

In this next section we will consider the Elkies codes evaluated on the Hermitian curve.

Definition 8 The **Hermitian Elkies code** with degree bound h , denoted as $E_h(H_{m^2})$ is defined as

$$E_h(H_{m^2}) = \{(f(Q_1), f(Q_2), f(Q_3), \dots, f(Q_\infty))\}$$

where f runs over the set of all rational functions f whose divisor of zeros and poles has the form

$$\operatorname{div}(f) = D - D'$$

where D and D' have only positive coefficients and $\deg(D), \deg(D') \leq h$.

We will concentrate on the cases $h = m$ and $h = m + 1$, in which all f will have the form

$$f = \frac{\alpha_1 x + \alpha_2 y + \alpha_3}{\beta_1 x + \beta_2 y + \beta_3}, \quad (1)$$

where $\alpha_i, \beta_i \in GF(m^2)$. For any rational function of this form, setting the numerator $\alpha_1 x + \alpha_2 y + \alpha_3$ equal to zero defines a line in the plane that intersects the Hermitian curve in exactly $m + 1$ points (possibly including Q_∞). Similarly, setting the denominator $\beta_1 x + \beta_2 y + \beta_3$ equal to zero yields another line intersecting the Hermitian curve at $m + 1$ points. This implies that the divisor of f is either

1. $\operatorname{div}(f) = D - D'$ where D, D' have degree $m + 1$, or
2. $\operatorname{div}(f) = D - D'$ where D, D' have degree m .

In the first case the lines defined by the numerator and denominator of f intersect at a point not on the Hermitian curve; in the second case they intersect at a point on the Hermitian curve. So by the definition above, in the first case, evaluating f gives a codeword in $E_{m+1}(H_{m^2})$, but not in $E_m(H_{m^2})$. In the second case, evaluating f gives a codeword in $E_m(H_{m^2})$. It follows from the definition that $E_m(H_{m^2}) \subset E_{m+1}(H_{m^2})$.

We notice that as was the case for the Elkies codes on the line, the $E_h(H_{m^2})$ codes are not linear because they are not closed under addition.

Theorem 3 Consider the code $E_{m+1}(H_{m^2})$ over the field $GF(m^2)$ and the subcode consisting of words obtained by evaluating functions with a fixed denominator. If this subcode is punctured at the positions corresponding to poles of the functions, the resulting code is a (possibly punctured) extended generalized Hermitian Goppa code with $a = m + 1$.

Proof: We know that $E_{m+1}(H_{m^2})$ is not linear, but if we look at codewords generated by functions with similar denominators then these subsets will have linear properties. Let $k(x, y) = \frac{g(x, y)}{d(x, y)}$ and $l(x, y) = \frac{h(x, y)}{d(x, y)}$ be the generating functions for words of $E_{m+1}(H_{m^2})$. Then we see that

$$k(x, y) + l(x, y) = \frac{g(x, y)}{d(x, y)} + \frac{h(x, y)}{d(x, y)} = \frac{g(x, y) + h(x, y)}{d(x, y)}$$

Because $g(x, y)$ and $h(x, y)$ are polynomials of total degree ≤ 1 and coefficients in $GF(m^2)$ then $g(x, y) + h(x, y)$ will also be a polynomial of total degree ≤ 1 with

coefficients in $GF(m^2)$. So the function $k(x, y) + l(x, y)$ also generates a word in $E_{m+1}(H_{m^2})$.

Furthermore we can divide $E_{m+1}(H_{m^2})$ into subcodes which are generated by evaluating the functions $f(x, y) \in \text{Span} \left\{ \frac{1}{d(x,y)}, \frac{x}{d(x,y)}, \frac{y}{d(x,y)} \right\}$. Let $\omega_i = \frac{1}{d(Q_i)}$ for all $Q_i \in H_{m^2}$ other than the zeroes of d . Let R_1, \dots, R_p be the points in H_{m^2} other than the zeroes of d . Then after puncturing at the pole positions a codeword $w \in E_{m+1}(H_{m^2})$ will have the form

$$(\omega_1 n(R_1), \omega_2 n(R_2), \dots, \omega_p n(R_p))$$

where $n(x, y) \in \text{Span}\{1, x, y\}$ is the numerator of f . If all the ω_i are equal, we obtain a codeword in a possibly punctured, extended generalized Hermitian Goppa code with $a = m + 1$. (The code may be an extended Goppa code because Q_∞ may be one of the points R_j). With the ω_i factors in w we get a word in a possibly punctured extended, generalized Goppa code. (The definition of a generalized Goppa code is analogous to the definition of a Reed Solomon code.) Hence every distinct denominator with 1 as the leading coefficient generates a unique Goppa code associated with $E_{m+1}(H_{m^2})$.

Corollary 1 $E_{m+1}(H_{m^2})$ can be divided into exactly $m^4 + m^2 + 1$ distinct possibly punctured, extended generalized Goppa codes.

Proof: Each possibly punctured extended generalized Goppa code contained in $E_{m+1}(H_{m^2})$ is associated with a specific denominator with 1 as the leading coefficient. If the denominator has both variables present then there will be m^2 for the coefficient of the second variable and for the coefficient of the constant. So for this case there are $(m^2)^2$ possible polynomial. If the denominator should only have one variable then there are just m^2 choices to chose the constant. Should the denominator be a constant then 1 is the only possible polynomial with 1 as the leading coefficient. Hence there are $m^4 + m^2 + 1$ distinct denominators or possible linear codes within $E_{m+1}(H_{m^2})$.

Corollary 2 Consider $E_m(H_{m^2})$ over the finite field $GF(m^2)$ and the subcode consisting of words obtained by evaluating functions with a fixed denominator. Then after puncturing this subcodes at the positions corresponding to poles of the functions, the resulting code is linear with dimention $k = 2$.

Proof: For $E_m(H_{m^2})$ the lines defined by the numerator and the denominator intersect at a point on the hermitian curve, therefor the divisor of f has a common *pole* and *zero*. To construct all evaluating functions f for $E_m H_{m^2}$ we consider all the combinations of functions that pass through a point $Q_i = (a_i, b_i)$ for all i .

We can easily see that the lines $x - a_i = 0$ and $y - b_i = 0$ meet at $Q_i = (a_i, b_i)$. This also implies that the lines $\alpha(x - a_i) + \beta(y - a_i) = 0$ for $\alpha, \beta \in GF(m^2)$ also pass through Q_i . Then we can define f as

$$f = \frac{\alpha(x - a_i) + \beta(y - b_i)}{\gamma(x - a_i) + \omega(y - b_i)}$$

for $\alpha, \beta, \gamma, \omega \in GF(m^2)$.

Therefore a codeword $w \in E_m(H_{m^2})$ is constructed by evaluating f at the points in H_{m^2} . $E_m(H_{m^2})$ as a whole is not linear since closure of addition does not hold.

Example. Let $m = 2$, $f(x) = \frac{y-b_i}{x-a_i}$, $g(x) = \frac{x-a_i}{y-b_i}$ so $f(x), g(x) \in F_{C(m)}$. Then

$$l(x) + s(x) = \frac{y - b_i}{x - a_i} + \frac{x - a_i}{y - b_i} = \frac{(y - b_i)^2 + (x - a_i)^2}{(x - a_i)(y - b_i)}$$

$l(x) + s(x)$ is not contained in $F_{C(m)}$.

If we consider codewords generated by functions with the same denominator then these makeup a linear code. Let $f(x, y) = \frac{\alpha_1(x-a_i) + \beta_1(y-b_i)}{den(Q_i)}$, $g(x, y) = \frac{\alpha_2(x-a_i) + \beta_2(y-b_i)}{den(Q_i)}$, $f(x, y), g(x, y) \in F_{C(m)}$ Then

$$\begin{aligned} f(x, y) + g(x, y) &= \frac{\alpha_1(x - a_i) + \beta_1(y - b_i)}{den(Q_i)} + \frac{\alpha_2(x - a_i) + \beta_2(y - b_i)}{den(Q_i)} \\ &= \frac{(\alpha_1 + \alpha_2)(x - a_i) + (\beta_1 + \beta_2)(y - b_i)}{den(Q_i)} \in F_{C(m)} \end{aligned}$$

Thus $E_m(H_{m^2})$ has linear codes associated to each distinct denominator, $den(Q_i)$, with 1 as the leading coefficient. Denominators with leading coefficients other than 1 can be considered multiples of a function with 1 as the leading coefficient, therefore these two polynomials would be the same code. We can easily see that each subcode is generated by evaluating all the functions, $f(x, y) \in span \left\{ \frac{x-a_i}{den(Q_i)}, \frac{y-b_i}{den(Q_i)} \right\}$. Therefore each subcode has rank of $k = 2$.

6 Conclusion

Throughout our research we looked at several types of codes and how their construction differed from one another. The Reed-Solomon codes served as a series of basic codes over a straight line, while Goppa codes formed a generalization of the Reed-Solomon codes. Constructing Elkie codes required a lot of time and patience, but our research was soon able to focus on its characteristics that would help on its non-linearity. We concluded that even though Elkie codes are nonlinear, they can be broken up into linear codes. With this characteristic, we could look into using the specific qualities of linear codes, such as generator and parity-check matrices, in hopes of finding ways to recognize an Elkie word, as well as trying to encode/decode the words. Due to our time constraint, our research remains open to many of these questions dealing with this unique type of code.

7 Future Work

The recent development of Elkie codes has allowed researchers to explore many avenues dealing with the codes construction and characteristics. In our research, we

explored several ideas that allowed us to better understand Elkies codes, such as their construction on algebraic curves and their grouping of linear subcodes. Eventhough a lot was learn, there still remains many questions unanswered. For future research, the question still remains on what is the best computable way to determine if a given word is an Elkies codeword. Also, the encoding method of Elkies codes is difficult since it is non-linear, so how could an easily computable 1-1 correspondence between information strings and codewords be set up. Lastly, Elkies codes contains a large group of automorphisms, so analyzing the groups of automorphisms that permute the points on the Hermitian curve and how their presence affect the coordinates of codewords in the Elkies codes remains as future research.

8 Acknowledgements

We would like to thank the following people:

- Dr. John Little, Jean Carlos Cortissoz, and Jesús Rodríguez for their time and assistance throughout our research experience.
- Dr. Herbert Medina, Dr. Ivelisse Rubio, and the SIMU staff for giving us the opportunity to participate in undergraduate research.
- Our fellow SIMU 2001 students for their support and encouragement during the program.
- The NSF, NSA, NASA, AMP, SACNAS, and the University of Puerto Rico-Humacao for their contributions to this program.

References

- [1] N. Elkies, *Excellent nonlinear codes from modular curves*, preprint 2001.
- [2] D.R. Hankerson, D.G. Hoffman, D.A. Leonard, C.C. Linder, K.T. Phelps, C.A. Rodger, J.R. Wall, *Coding Theory and Cryptography*, Second Edition. Marsal Dekker, New York, 2000.
- [3] F.J. MacWilliams & N.J.A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland Mathematical Library, v. 16. Elsevier Science Publishers B.V., New York, 1997.
- [4] O. Pretzel, *Codes and Algebraic Curves*, Oxford University Press, New York, 1998.
- [5] J. Walker, *Codes and Curves*, Student Mathematical Library, v. 7. IAS/Park City Mathematical Subseries. American Mathematical Society, Providence, RI; Institute for Advanced Study (IAS), Princeton, NJ, 2000.

Appendix A: Algorithms

```
GenGF:=proc(h::polynom,p::integer)
# Construct the elements of the finite field GF(p^r) using an
# irreducible polynomial h(x).
```

```
local r,m,i,beta,g,k,lista;
```

```
beta=RootOf(h);
r=degree(h);
m=p^r-2;
lista=[0];
for i from 0 to m do
    beta[i]=rem(x^i,h,x) mod p;
    lista=[op(lista),beta[i]];
    lista=subs(x=beta,lista);
end do;
RETURN(lista)
end:
```

```
-----
AllPoints:=proc(m::integer,h::polynom,p::integer)
```

```
# Computes all the solution points for the Hermitian curve
# with parameter m over GF(p^r) using h(x).
```

```
local list1,list,i,j,k,r,beta;
```

```
beta=RootOf(h);
r=degree(h);
list1=[0];
list=[];
for k from 0 to p^r-2 do
    list1=[op(list1),beta^k];
end do;
for i to p^r do
    for j to p^r do
if Normal(list1[i]^(m+1)) mod p = Normal(list1[j]^m) mod p +
    Normal(list1[j]) mod p then
        list:=op(list,vector([list1[i],list1[j]]));
        end if;
    end do;
end do;
RETURN(list)
end:
```

```

AllFunctions:=proc(a::integer,m::integer)

# Generates the functions  $x^r*y^s$  such that  $mr+(m+1)s \leq a$ .

local xbound,ybound,list,i,j;

xbound=trunc(a/m);
ybound=trunc(a/(m+1));
list=[];
for i from 0 to xbound do
  for j from 0 to ybound do
    if (m*i)+(m+1)*j <= a then
      list:=[op(list),x^i*y^j];
    end if;
  end do;
end do;
RETURN(list);
end:
-----

CodeWords:=proc(a::integer,m::integer,h::polynom,p::integer)
# Produces the Goppa codewords by evaluating the functions at
# the solution points.

local points,func,i,j,n,r,xval,yval,list,word,mlist,t,beta;

beta:=RootOf(h);
points:=AllPoints(m,h,p);
func:=AllFunctions(a,m);
n:=nops(points);
r:=degree(h);
t:=nops(func);
list=[];
for i to t do
  word=[];
  for j to n do
    xval:=(points[j])[1];
    yval:=(points[j])[2];
    word:=[op(word),subs({x=xval,y=yval},func[i])];
  end do;
list:=[op(list),word];
mlist:=matrix(list);
end do;
RETURN(eval(mlist))
end:
-----

```

```

Coefficients:=proc(p::integer,poly::polynom)
# Generates all the coefficients of the functions that will
# be evaluated at the solution points to get the Elkies codes.

local GF,nGF,list,i,k,beta,r;

beta:=RootOf(poly);
r:=degree(poly);
GF:=[0,seq(beta^i,i=0..p^r-2)];
nGF:=nops(GF);
list:=[];
for i to nGF do
  for k to nGF do
    list:=[op(list),vector([GF[i],GF[k]])];
  end do;
end do;
RETURN(list)
end:
-----

PolyOne:=proc(p::integer,poly::polynom)
# Produces all the polynomials of the type (x+alpha),
# where alpha is in GF(p^r).

local nl,listpoly,j,poly1,c,cf,beta,field,r,q;

beta:=RootOf(poly);
r:=degree(poly);
listpoly:=[];
field:=[0,seq(beta^i,i=0..p^r-2)];
q:=nops(field);
for j to q do
  poly1:=x+field[j];
  listpoly:=[op(listpoly),poly1];
end do;
RETURN(eval(listpoly))
end:
-----

AllPolyOne:=proc(h::integer,q::integer,p::integer,poly::polynom)
# Creates all the polynomials of the type: alpha1, alpha1(x+alpha2),
# alpha1*(1/x+alpha3), and alpha1*(x+alpha2/x+alpha3).

local pol,np,r,cf,ncf,v,i,j,k,l,list,f,f1,f2,f3,beta;

beta:=RootOf(poly);
pol:=Polys(h,q,p,poly);

```

```

np:=nops(pol);
r:=degree(poly);
cf:=seq(beta^i,i=0..p^r-2);
ncf:=nops(cf);
list:=[];
for v to ncf do
  f3:=cf[v];
  list:=op(list),f3;
end do;
for i to np do
  for k to ncf do
    f1:=cf[k]*pol[i];
    f2:=cf[k]*(1/pol[i]);
    list:=op(list),f1,f2;
  end do;
  for j to np do
    if i <> j then
      for l to ncf do
        f:=cf[l]*pol[i]/pol[j];
        list:=op(list),f;
      end do;
    end if;
  end do;
end do;
RETURN(eval(list))
end:
-----
RSSub:=proc(p::integer,poly::polynom,k::integer)
# Computes all the independent polynomials with
# similar denominator (x+k).

local sp,nsp,list,r,lim,cf,den,i,f;

sp:=SelectPoly(p,poly);
nsp:=nops(sp);
list:=[];
r:=degree(poly);
lim:=p^r;
cf:=seq(beta^i,i=0..p^r-2);
den:=x+cf[k];
for i to nsp do
  if k>lim then
    print ("denominator constant not in GF" );
  else
    f:=sp[i]/den;

```

```

        list:=[op(list),f];
    end if;
end do;
RETURN(list)
end:
-----
SubElkGenMat:=proc(p::integer, poly::polynom)
# Produces a list of generating matrices for the subcodes
# of an Elkies Code of degree bound h=1.

local cf,ncf,list2,word1,word2,den,f1,f2,j,i,f1e,f2e,mat,r,beta,s;

beta:=RootOf(poly);
r:=degree(poly);
cf:=[seq(beta^i,i=0..p^r-2),0,infinity];
ncf:=nops(cf);
list2:=[matrix([[seq(1,j=1..ncf)],Normal(cf) mod p]),
matrix([[seq(1,j=1..ncf)],[seq(Normal(1/cf[j]) mod p,j=1..p^r-1),
infinity,0]])];
for i to ncf-2 do
    word1:=[];
    word2:=[];
    den:=x+cf[i];
    f1:=1/den;
    f2:=x/den;
    lprint (f1);
    lprint (f2);
    for j to ncf do
        if cf[j]<>infinity then
            s:=Normal(subs(x=cf[j],den)) mod p;
            if s=0 then
                f1e:=infinity; f2e:=infinity;
            else
                f1e:=Normal(subs(x=cf[j],f1)) mod p;
                f2e:=Normal(subs(x=cf[j],f2)) mod p;
            end if;
            word1:=[op(word1),f1e];
            word2:=[op(word2),f2e];
        else
            f1e:=0;
            f2e:=1;
            word1:=[op(word1),f1e];
            word2:=[op(word2),f2e];
        end if;
    end do;
end do;

```

```

        mat:=matrix([word1,word2]);
        list2:=[op(list2),eval(mat)];
end do;
RETURN(list2)
end:
-----
ElkCode:=proc(p::integer,poly::polynom,h::integer)
# Creates the codewords of an Elkies Code on a straight line.

local allpol,nap,points,epoints,nep,i,j,m,q,list,r,c,code,
      n,d,beta,sublist,nlist;

beta:=RootOf(poly);
allpol:=AllPolys(p,poly);
nap:=nops(allpol);
r:=degree(poly);
points:=[seq(beta^i,i=0..$p^r-2$)];
epoints:=[op(points),0,infinity];
nep:=nops(epoints);
list:=[];
sublist:={seq(Normal(beta^i) mod p = beta^i,i=0..p^r-2)};
for i to nap do
  code:=[];
  n:=numer(allpol[i]);
  d:=denom(allpol[i]);
  for j to nep-1 do
    if Normal(subs(x=epoints[j],denom(allpol[i]))) mod p = 0 then
      c:=infinity;
    else c:=Normal(subs(x=epoints[j],allpol[i])) mod p; end if;
    c:=subs(sublist,c);
    code:=[op(code),c];
  end do;
  for q from 0 to h do
    if degree(n)=degree(d) and degree(n) = q then
      c:=Normal(coeff(n,x,q) mod p; end if;
    end do;
    if degree(n)<degree(d) then c:=0; end if;
    if degree(n)>degree(d) then c:=infinity; end if;
    c:=subs(sublist,c);
    code:=[op(code),c];
    list:=[op(list),eval(code)];
  end do;
RETURN(eval(list))
end:

```