Implementing Generalized Reed-Solomon Codes and a Cyclic Code Decoder in GUAVA
Jason McGowan
4-4-2005

Contents
1 Problem 1
2 Introduction 2
3 Algorithms 6
4 Examples 8
5 GAP Code 9
6 Conjecture 15

1 Problem

1. Implement evaluation codes. (Algorithm 1)

2. Write a decoding algorithm for Generalized Reed-Solomon codes. (Algorithm 2)

3. Write a decoding algorithm for cyclic codes. (Algorithm 3)

Background Information: GAP is a computer algebra program whose open source kernel is written in the C programming language. GUAVA is a package for error-correcting block codes in GAP. Neither GAP nor GUAVA contains a program for decoding Generalized Reed-Solomon codes or cyclic codes, two very popular families of codes. Our work will greatly speed the decoding algorithms for several types of codes in the next release of the GUAVA package.
2 Introduction

A linear code of length \( n \) is a subspace of \( GF(q)^n \) for some prime power \( q \) and some \( n > 1 \). Let \( C \) be a linear code of length \( n \) over \( \mathbb{F} = GF(q) \). If \( q = 2 \) then the code is called binary. We assume that \( \mathbb{F}^n \) has been given the standard basis \( e_1 = (1, 0, \ldots, 0) \in \mathbb{F}^n, e_2 = (0, 1, 0, \ldots, 0) \in \mathbb{F}^n, \ldots, e_n = (0, 0, \ldots, 0, 1) \in \mathbb{F}^n \). The dimension of \( C \) is denoted \( k \), so the number of elements of \( C \) is equal to \( q^k \).

Another important parameter associated to the code is the number of errors which it can, in principle, correct. For this notion, we need to introduce the Hamming metric. For any two \( x, y \in \mathbb{F}^n \), let \( d(x, y) \) denote the number of coordinates where these two vectors differ:

\[
d(x, y) = |\{1 \leq i \leq n \mid x_i \neq y_i\}|.
\]

This is called the Hamming distance or Hamming metric. Define the weight \( w \) of \( v \) to be the number of non-zero entries of \( v \). Note, \( d(x, y) = w(x−y) \) because the vector \( x−y \) has non-zero entries only at locations where \( x \) and \( y \) differ. The smallest distance between distinct codewords in a linear code \( C \) is the minimum distance of \( C \):

\[
d = d(C) = \min_{c \in C, c \neq 0} d(0, c)
\]

(for details see [HILL] Theorem 5.2).

A linear code \( C \) of length \( n \) and dimension \( k \) over \( \mathbb{F} \) has a basis of \( k \) vectors of length \( n \). If those vectors are arranged as rows of a matrix \( G \), then we call the \( k \times n \) matrix \( G \) a generator matrix for \( C \).

**Example 1** The following matrix is a generator matrix for a code of length 11, dimension 8, and minimum distance 4 over \( GF(11) \).

\[
G = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7 & 2 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 8 & 3 & 10 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 2 & 7 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 8 & 8 & 5 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 10 & 3 & 8 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 4 & 5 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 5 & 4 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 5 & 8 & 8 
\end{pmatrix}
\]
Now we define a general class of codes which, as part of this project, were implemented in GUAVA. Let \( R = \mathbb{F}[x_1, ..., x_d] \) denote the polynomial ring in \( d \) variables \( x_1 \) through \( x_d \). Let \( K = \mathbb{F}(x_1, ..., x_d) \) denote the field of rational functions in these \( d \) variables. Let \( V \) be a finite dimensional subspace of \( K \) and choose a subset \( E \) of points of \( \mathbb{F}^d \) disjoint from the poles of the functions in \( V \) (so all the functions \( f \in V \) are defined at the points \( p \in E \)). Construct a linear code \( C \) by evaluating basis vectors of \( V \) at the points of \( E \) as follows. Choose \( f_1, ..., f_k \in K \) which are basis vectors for \( V \) and pick \( E = \{ p_1, ..., p_n \} \subset \mathbb{F}^d \) as above. Denote by \( \text{Eval}_E(f) = (f(p_1), f(p_2), ..., f(p_n)) \) the evaluation map \( \text{Eval}_E : K \to (\mathbb{F} \cup \{\infty\})^d \). Define the associated evaluation code \( C \) by means of the \( k \times n \) generator matrix,

\[
G = (f_i(p_j))_{1 \leq j \leq n, 1 \leq i \leq k}.
\]

This is the code spanned by the vectors \( \text{Eval}_E(f_i), 1 \leq i \leq k \).

**Example 2** Let \( n \) be greater than \( q \). Let \( R = \mathbb{F}[x] \) denote the polynomial ring in \( x \). Let \( V = R_k \) denote the vector space of polynomials with coefficients in \( \mathbb{F} \) of degree less than \( k \). Let \( E \) denote a subset of \( n \) elements of \( \mathbb{F} \) (since \( n > q \) this makes sense). Choose as a vector space basis of \( V \) the powers of \( 1, x, x^2, ..., x^{k-1} \). In this case the evaluation code has generator matrix,

\[
G = (p_j^{i-1})_{1 \leq j \leq n, 1 \leq i \leq k}.
\]

This is a generalized Reed-Solomon code of length \( n \) and dimension \( k \) over \( \mathbb{F} \).

**Lemma 3** Let \( C \) denote a generalized Reed-Solomon code as in above example. \( C \) has minimum distance \( d = n - k + 1 \).

**Proof** Suppose a codeword \( c \) has weight less than \( n - k + 1 \). Associated to \( c \) is a polynomial \( f(x) \) of degree less than \( k \) with the property that \( c = (f(x_1), f(x_2), ..., f(x_n)) \). If the weight of \( c \) is less than \( n - k + 1 \) then \( f(x) \) must have greater than \( n - (n - k + 1) \) zeros. Therefore \( f \) has greater than \( k - 1 \) zeros. This contradicts the fact that \( f \) has at most \( k - 1 \) zeros. \( \square \)

Therefore all generalized Reed-Solomon codes are minimum distance separable.
Definition 4 A linear code $C$ of length $n$ is a **cyclic code** if whenever $c = (c_1, ..., c_n)$ is a codeword then so is its cyclic shift $c' = (c_2, ..., c_n, c_1)$.

Example 5 Consider the binary code $C$ with generator matrix

$$G = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

Clearly these four rows $g_1, g_2, g_3, g_4$ are obtained from the previous by a shift to the right. Also notes the shift of $g_4$ to the right is equal to $g_5 = g_1 + g_3 + g_4$. The shift of $g_5$ to the right is $g_6 = g_1 + g_2 + g_3$. And the shift of $g_6$ is $g_7 = g_2 + g_3 + g_4$. The shift of $g_7$ is $g_1$. Therefore, the linear code generated by $G$ is invariate under shifts to the right. Therefore $C$ is a cyclic code.

Cyclic codewords are conveniently represented as polynomials modulo $x^n - 1$. In fact, if $c = (c_1, ..., c_n)$ then let

$$c(x) = c_1 + c_2 x + ... + c_n x^{n-1}$$

denote the associated **codeword polynomial**. In this notation the cyclic shift $c' = (c_2, ..., c_n, c_1)$ of $c$ corresponds to the polynomial $xc(x)$ (mod $x^n - 1$). In other words cyclic shifts correspond to multiplication by $x$. Since cyclic shifts leave cyclic codes invariant, multiplication by any power of $x$ modulo $x^n - 1$ corresponds to a codeword in $C$. Since $C$ is a linear code, the sum of any two such codeword polynomials is another codeword polynomial. Therefore, in fact, the product of any codeword polynomial times any polynomial in $x$ modulo $x^n - 1$ is another codeword polynomial.

Denote by $R_n$ the ring of polynomials with coefficients in $\mathbb{F}$ modulo $x^n - 1$:

$$R_n = \mathbb{F}[x]/(x^n - 1).$$

(3)

Define an **ideal** $I$ of $R_n$ to be any subset of $R_n$ closed under addition and multiplication by an arbitrary element of $R_n$:

- If $f, g \in I$ then $f + g \in I$, and
- If $f \in I$ and $r \in R_n$ then $rf \in I$. 


In other words an ideal in \( R_n \) is simply a subset closed under addition and multiplication by an arbitrary polynomial modulo \( x^n - 1 \). In particular, the collection of codeword polynomials associated to a cyclic code is an ideal of \( R_n \).

**Lemma 6** There is natural one-to-one correspondence between cyclic codes of length \( n \) over \( \mathbb{F} \) and ideals of \( R_n \).

This can be found in any book on coding theory, for example MacWilliams and Sloane [MS].

In fact GUAVA allows you to easily pass back and forth between codewords as vectors and codewords as polynomials.

In order to define the generator polynomial of a cyclic code we need the following mathematical fact.

**Lemma 7** Every ideal \( I \) of \( R_n \) is of the form \( g(x)R_n \). In other words every element of \( I \) is a multiple of \( g(x) \) for some polynomial \( g(x) \) in \( R_n \).

Ideals which are of the form \( I = g(x)R_n \) are called **principal ideals** and \( g(x) \) is called a **generator** of the ideal \( I \).

**Proof** Suppose not. Let \( s(x) \) be a non-zero element in \( I \) of smallest degree. Pick an arbitrary non-zero element \( f(x) \) in \( I \). By the division algorithm, we can write \( f(x) = q(x)s(x) + r(x) \) where \( q \) and \( r \) are polynomials and the degree of \( r(x) \) is strictly less than the degree of \( s(x) \). Notice that \( r(x) = f(x) - q(x)s(x) \) belongs to \( I \) by definition. This contradicts the assumption that \( s(x) \) has smallest degree unless \( r(x) = 0 \). Therefore every element of \( I \) is a multiple of \( s(x) \). Take \( g(x) = s(x) \). ☐

**Definition 8** Let \( C \) be a cyclic code of length \( n \). Let \( I \) be the ideal corresponding to \( C \) by Lemma 6. We call \( g(x) \) a **generator polynomial** of \( C \) if it is a generator of \( I \).

**Example 9** We continue with Example 5. Let \( g(x) = 1 + x^2 + x^3 \). This is the codeword polynomial associated to the top row of the generator matrix. \( g(x) \) is the generator polynomial of the cyclic code \( C \). Note that \( x^7 - 1 = (x + 1)(x^3 + x^2 + 1)(x^5 + x + 1) \).

More generally we have the following result

**Lemma 10** The generator polynomial always divides \( x^n - 1 \).
Proof By the division algorithm $x^n - 1 = q(x)g(x) + r(x)$ for some quotient $q(x)$ and some remainder $r(x)$ of degree less than $g(x)$. In $R_n$ this means that $r(x) = -q(x)g(x)$. So $r(x)$ corresponds to a codeword since $g(x)$ is the generator polynomial. This is impossible since $g(x)$ has minimal degree of any codeword in $C$ unless $r(x) = 0$. □

3 Algorithms

Recall $F = GF(q)$.

One common method of constructing a linear code is by constructing its generator matrix from the values of certain functions at certain points. Sometimes this construction adds extra structure to the code which aids in implementing fast decoding algorithms. The algorithm below assumes that you have a list $L$ of rational functions in $r$ variables and a list $E$ of $n$ points in $F$.

Algorithm 1 (Evaluation Code)

1. Input: $E$ is a list of $n$ points in $F^r$, $L$ is a list of $m$ rational functions in $r$ variables. $R$ is the ring of multivariate polynomials in $r$ variables of which $L$ is a subset.

2. For $1 \leq i \leq n$ and $1 \leq j \leq m$ let $g_{i,j} = f_j(p_i)$, where $p_i \in E$ and $f_j \in L$. Let $C$ be the code with generator matrix $G = (g_{i,j})$. This is a $[n, \leq m]$ code over $F$.

The decoding of generalized Reed-Solomon codes relies on the algebraic structure of the code. Codewords are values of a polynomial of degree less than $k$. The rough idea is that if enough values are known, then the polynomial can be reconstructed (think of the Lagrange interpolation formula).

Let $r = c + e = (r_1, ..., r_n)$ be a received word where $c = (f(p_1), ..., f(p_n))$ is a codeword and the weight of $e$ is less than $\tau = [(d - 1)/2]$, where $[...]$ denotes the integer part. The interpolating polynomial $Q(x, y) = Q_0(x) + yQ_1(x)$ is defined to be a polynomial of two variables having the following three properties:

1. $Q(p_i, r_i) = 0$, for all $i$ with $1 \leq i \leq n$,

2. $\text{deg}(Q_0) \leq n - 1 - \tau,$

3. \( \deg(Q_1) \leq n - \tau - k \)

First we show that an interpolating polynomial always exists.

**Lemma 11** There always exists a non-zero polynomial \( Q(x, y) \) satisfying the conditions above.

**Proof** By condition 2 we can write

\[
Q_0(x) = a_0 + a_1x + \ldots + a_{n-1-\tau}x^{n-2-\tau},
\]

and by condition 3 we can write

\[
Q_1(x) = b_0 + b_1x + \ldots + b_{n-\tau-k}x^{n-\tau-k-1}.
\]

So we have \( 2n-2\tau-k-1 \) degrees of freedom in choosing the coefficients for \( Q \). Condition 1 imposes \( n \) constraints on these coefficients, therefore giving rise to a linear system of \( n \) equations in \( 2n-2\tau-k+1 \) unknowns. By Lemma 3, \( \tau = [(d-1)/2] = [(n-k)/2] \). So \( 2n-2\tau-k+1 \geq 2n-2(n-k)/2-k+1 = n+1 \). Therefore, this system has a solution. □

Why is it that computing \( Q(x, y) \) will help us to find the polynomial \( f(x) \) that gave rise to the codeword \( c \) that we’re looking for? Here’s the intuitive explanation. Since there aren’t too many errors in the received word, it follows that “usually” \( f(p_i) = r_i \). Therefore, by Property (1) above, \( Q(p_i, f(p_i)) \) is “usually” 0. But notice that Properties (2) and (3) above, \( Q(x, f(x)) \) is a polynomial in \( x \) of degree at most \( n - \tau - 2 \). So if the number \( x = p_i \) which are roots of \( Q(x, f(x)) \) exceeds \( n - \tau - 2 \), then \( Q(x, f(x)) \) must be equal to 0 identically. But if \( Q(x, f(x)) = 0 \), then it is easy to solve for \( f(x) \): \( Q_0(x) + f(x)Q_1(x) = 0 \) implies \( f(x) = \frac{Q_0(x)}{Q_2(x)} \).

**Algorithm 2** (Generalized Reed-Solomon Decoder)

1. **Input:** \( E = \{p_1, \ldots, p_n\} \) is a list of \( n \) points in \( F \), \( k < n \) is an integer.
   **Output:** A codeword in \( C \) close to \( v \) or “fail”.

2. Let \( Q_0(x) = a_0 + a_1x + \ldots + a_{n-1-\tau}x^{n-2-\tau} \) and \( Q_1(x) = b_0 + b_1x + \ldots + b_{n-\tau-k}x^{n-\tau-k-1} \). Property (1) above gives \( n \) equations
   \[
   Q(p_i, r_i) = Q_0(p_i) + r_i Q_1(p_i)
   = a_0 + a_1p_i + \ldots + a_{n-1-\tau}p_i^{n-2-\tau} + r_i(b_0 + b_1p_i + \ldots + b_{n-\tau-k}p_i^{n-\tau-k-1})
   \]
   in \( 2n - 2\tau - k + 1 \) unknowns \( \{a_0, \ldots, a_{n-1-\tau}, b_0, \ldots, b_{n-\tau-k}\} \). Solve these equations for these unknowns.
3. Compute using the previous step \( Q(x, y) = Q_0(x) + yQ_1(x) \).

4. Let \( f(x) = -\frac{Q_0(x)}{Q_1(x)} \). Return \( c = (f(p_1), \ldots, f(p_n)) \).

Consider a cyclic code \( C \) with generator polynomial \( g(x) \) of length \( n \) over \( \mathbb{F} \) and a received word \( v \) in \( \mathbb{F}^n \). Let \( v(x) \) denote the polynomial associated to the received word \( v \). Consider the “syndrome polynomial” \( s(x) = v(x) \pmod{g(x)} \). If the degree of \( s(x) \) is less than \( [(d-1)/2] \) then a) the polynomial \( c(x) = v(x) - s(x) \) is within Hamming distance \( (d - 1)/2 \) of the received polynomial \( v(x) \), and b) \( c(x) \) is associated to a codeword \( c \) in \( C \). Therefore \( c \) is the codeword that the minimum distance algorithm would return for decoding \( v \).

**Algorithm 3 (Cyclic Decoder)**

1. Input: \( C \) is a cyclic code with generator polynomial \( g(x) \) of length \( n \) over \( \mathbb{F} \) and a received word \( v \) in \( \mathbb{F}^n \).
   
   Output: A codeword in \( C \) close to \( v \) or “fail”.

2. Compute the polynomial \( v(x) \) associated to the received word \( v \).

3. For each \( i \) from 1 to \( n - 1 \), compute the syndrome polynomials \( s_i(x) = x^i v(x) \pmod{g(x)} \). Let \( d_i \) denote the degree of \( s_i \).

4. Find the first \( d_i \) which is less than \( [(d-1)/2] \). Let \( e(x) = x^{n-i} s_i(x) \pmod{x^n - 1} \). Let \( c(x) = v(x) - e(x) \).

5. Convert \( c(x) \) to a codeword \( c \) and return \( c \).

The algorithm will never return fail if the number of errors is less than \( [(d-1)/2] \).

**4 Examples**

In the following two examples we consider a cyclic code \( C \) of length 10 over the field with three elements \( GF(3) \). Let \( c \in C \) denote a random codeword. We create an error in the first coordinate of \( c \) and use the method of Algorithm 3 to decode this incorrect received word.
### example - stays in 1st loop

CCs:=CyclicCodes(10,GF(3));;
C:=CCs[2];
c:=Random(C);
e:=Codeword(Z(3)*[1,0,0,0,0,0,0,0,0,0,0],GF(3));
CyclicDecodeword(c+e,C);

gap> CCs:=CyclicCodes(10,GF(3));;
gap> C:=CCs[2];
a cyclic [10,9,1..2]1 enumerated code over GF(3)
gap> c:=Random(C);
[ 2 1 0 1 1 1 2 0 2 ]
gap> e:=Codeword(Z(3)*[1,0,0,0,0,0,0,0,0,0,0],GF(3));
[ 2 0 0 0 0 0 0 0 0 0 0 ]
gap> CyclicDecodeword(c+e,C);
[ 2 1 0 1 1 1 2 0 2 ]

#### example - goes into 2nd loop

CCs:=CyclicCodes(15,GF(3));;
C:=CCs[6];
MinimimDistance(C);
c:=Random(C);
e:=Codeword(Z(3)*[1,0,0,0,0,0,0,0,0,0,0,0,0,0,1],GF(3));
CyclicDecodeword(c+e,C);

## 5 GAP Code

ValueExtended := function(f, vars, P)
  return Value(NumeratorOfRationalFunction(f)*vars[1]^0, vars, P)
  * Value(DenominatorOfRationalFunction(f)*vars[1]^0, vars, P)^(-1);
end;

ValueExtended := function(f, vars, P)
  return Value(NumeratorOfRationalFunction(f)*vars[1]^0, vars, P)
  * Value(DenominatorOfRationalFunction(f)*vars[1]^0, vars, P)^(-1);
end;

# P is a list of n points in F^r
# L is a list of rational functions in r variables
# EvaluationCode returns the image of the evaluation map f->[f(P1),...,f(Pn)]
# The output is the code whose generator matrix has rows (f(P1)...f(Pn)) where
# f is in L and P={P1,...,Pn}
EvaluationCode := function(P, L, R)
  local i, G, n, C, j, k, vars, varsd, vars, F;
  vars := IndeterminatesOfPolynomialRing(R);
  varsd := IndeterminatesOfPolynomialRing(R);
F:=CoefficientsRing(R);
n:=Length(P);
k:=Length(L);
G:=ShallowCopy(NullMat(k,n,F));
for i in [1..k] do
  for j in [1..n] do
    G[i][j]:=ValueExtended(L[i],vars,P[j]);
  od;
od;
C:=GeneratorMatCode(G,F);
C!.GeneratorMat:=ShallowCopy(G);
return C;
end;

GeneralizedReedSolomonCode:=function(P,k,R)
local p, L, vars, i, F, f, x, C, R0, P0, G, j, n;
n:=Length(P);
F:=CoefficientsRing(R);
G:=NullMat(k,n,F);
vars:=IndeterminatesOfPolynomialRing(R);
x:=vars[1];
L:=List([0..(k-1)],i->(x^i));
for i in [1..k] do
  for j in [1..n] do
    G[i][j]:=Value(L[i],vars,[P[j]]);
  od;
od;
C:=GeneratorMatCode(G,F);
C!.GeneratorMat:=ShallowCopy(G);
return C;
end;

GeneralizedReedMullerCode:=function(Pts,r,F)
## Pts are points in F^d
## for usual GRM code, take
## pts:=Cartesian(List([1..d],i->Elements(F)));
## for some d>1.
## r is the degree of the polys in x1, ..., xd
##
local q, n, pts, row, B0, L, exps, Ld, i, x, e, d;
q:=Size(F);
d:=Length(Pts[1]);
L:=[0..Minimum(q-1,r)];
Ld:=Cartesian(List([1..d],i->L));
exps:=Filtered(Ld,x->Sum(x)<=r);
n:=Size(pts);
B0:=[ ];
for e in exps do
    row:=List(Pts,t->Product([1..d],i->t[i]^e[i]));
    Add(B0,row);
od;
B:=AsList(B0);
C:=GeneratorMatCode(One(F)*B,F);
return C;
end;

CyclicDecodeword:=function(w,C)
    local d, g, wpol, s, ds, cpol, cc, c, i, m, e, x, n, ccc, r;
    if not(IsCyclicCode(C)) then
        Error("\n\n Code must be cyclic\n\n");
    fi;
    n:=WordLength(C);
    d:=MinimumDistance(C);
    g:=GeneratorPol(C);
    x:=IndeterminateOfUnivariateRationalFunction(g);
    wpol:=PolyCodeword(w);
    s:=wpol mod g;
    ds:=DegreeOfLaurentPolynomial(s);
    if ds<=Int((d-1)/2) then
        cpol:=wpol-s;
        cc:=CoefficientsOfUnivariatePolynomial(cpol);
        r:=Length(cc);
        ccc:=Concatenation(cc,List([1..(n-r)],i->0*cc[1]));
        c:=Codeword(ccc);
        return c;
    fi;
    for i in [1..(n-1)] do
        s:=x^i*wpol mod g;
        ds:=DegreeOfLaurentPolynomial(s);
        if ds<=Int((d-1)/2) then
            m:=i;
            e:=x^{(n-m)}*s mod (x^n-1);
            cpol:=wpol-e;
            cc:=CoefficientsOfUnivariatePolynomial(cpol);
            r:=Length(cc);
            ccc:=Concatenation(cc,List([1..(n-r)],i->0*cc[1]));
            c:=Codeword(ccc);
        fi;
    od;
end;
return c;
fi;
od;
return "fail";
end;

add_col_mat:=function(M,N)
#N is a matrix with same rowdim as M
#the fcn adjoins N to the end of M
local i,j,S,col,NT;
col:=MutableTransposedMat(M); #preserves M
NT:=MutableTransposedMat(N); #preserves N
for j in [1..DimensionsMat(N)[2]] do
  Add(col,NT[j]);
od;
return MutableTransposedMat(col);
end;

add_row_mat:=function(M,N)
#N is a matrix with same coldim as M
#the fcn adjoins N to the bottom of M
local i,j,S,row;
row:=ShallowCopy(M);#to preserve M;
for j in [1..DimensionsMat(N)[1]] do
  Add(row,N[j]);
od;
return row;
end;

block_matrix:=function(L)
#L is an array of matrices of the form
#[[M1,...,Ma],[N1,...,Na],...,[P1,...,Pa]]
#returns the associated block matrix
local A,B,i,j,m,n;
n:=Length(L[1]);
m:=Length(L);
A:=[[]];
if n=1 then
  if m=1 then return L[1][1]; fi;
  A:=L[1][1];
  for i in [2..m] do
    A:=add_row_mat(A,L[i][1]);
  od;
  return A;
fi;
for j in [1..m] do
    A[j]:=L[j][1];
    od;
for j in [1..m] do
    for i in [2..n] do
        A[j]:=add_col_mat(A[j],L[j][i]);
        od;
    od;
B:=A[1];
for j in [2..m] do
    B:= add_row_mat(B,A[j]);
    od;
return B;
end;

DiagonalPower:=function(r,j)
    local R,n,i;
    n:=Length(r);
    R:=DiagonalMat(List([1..n],i->r[i]^j));
    return R;
end;

# Input: Xlist=[x1,..,xn], l = highest power
# Output: Vandermonde matrix (xi^j)
#
VandermondeMat:=function(Xlist,x)
    local V,n,i,j;
    n:=Length(Xlist);
    V:=List([1..(x+1)],j->List([1..n],i->Xlist[i]^(j-1)));
    return TransposedMat(V);
end;

# Input: Xlist=[x1,..,xn], l = highest power,
# L=[h_1,...,h_ell] is list of powers
# r=[r1,...,rn] is received vector
# Output: Computes matrix described in Algor. 12.1.1 in [JH]
#
LocatorMat:=function(r,Xlist,L)
    local a,j,b,ell;
    ell:=Length(L);
    a:=List([1..ell],j->DiagonalPower(r,(j-1))*VandermondeMat(Xlist,L[j]));
    b:=List([1..ell],j->[1,j,a[j]]);
# return BlockMatrix(b,1,ell);
  return (block_matrix([a]));
end;

# Compute kernel of matrix in alg 12.1.1 in [JH].
# Choose a basis vector in kernel.
# Construct the polynomial Q(x,y) in alg 12.1.1.
#
ErrorLocatorPair:=function(r,Xlist,L)
  local a,j,b,q,e,Q,i,lengths,ker,ell;
  ell:=Length(L);
  e:=LocatorMat(r,Xlist,L);
  ker:=NullspaceMat(TransposedMat(e));
  if ker=[] then Print("Decoding fails.
"); return []; fi;
  q:=ker[1];
  Q:=[ ];
  lengths:=List([1..Length(L)],i->Sum(List([1..i],j->1+L[j])));
  Q[1]:=List([1..lengths[1]],j->q[j]);
  for i in [2..Length(L)] do
    Q[i]:=List([(lengths[i-1]+1)..lengths[i]],j->q[j]);
  od;
  return Q;
end;

CoefficientToPolynomial:=function(coeffs,R)
  local p,i,j, lengths, F,xx;
  xx:=IndeterminatesOfPolynomialRing(R)[1];
  F:=Field(coeffs);
  p:=Zero(F);
  lengths:=List([1..Length(coeffs)],i->Sum(List([1..i],j->1+coeffs[j])));
  for i in [1..Length(coeffs)] do
    p:=p+coeffs[i]*xx^(i-1);
  od;
  return p;
end;

# Input: List L of coefficients ell, R = F[x]
# Output: list of polynomials Q_i as in Algor. 12.1.1 in [JH]
6 Conjecture

There is a simple relationship between finding the minimum weight codeword of a linear code and decoding a received word. Here is an illustration: Let $G$ denote a generator matrix of a linear code $C$. Let $\mathbf{v} = \mathbf{c} + \mathbf{e}$ denote a received word with error vector $\mathbf{e}$ and codeword $\mathbf{c}$. Create the new code $C'$ with generator matrix

$$G' = \begin{pmatrix} G \\ \mathbf{v} \end{pmatrix}.$$

The minimum weight codeword of this code is the error vector $\mathbf{e}$ provided $\mathbf{e}$ has less than $[(d - 1)/2]$ non-zero coordinates. Therefore if you can compute the minimum weight codeword of $C'$ then you can decode any received word for $C$. 

15
The following conjecture tries to reverse this idea and ask if the decoding algorithm for a cyclic code can help one determine the weight of a minimum weight codeword. If one could reverse this idea, this would have the advantage of speeding up the algorithm for determining the minimum weight codeword. (The decoding algorithm for the cyclic code is relatively fast compared to the algorithm for finding the minimum weight codeword.)

**Conjecture 1**

1. **Input:** \( C \) is a cyclic code with generator polynomial \( g(x) \) of length \( n \) over \( \mathbb{F} \) and a received word \( v \) in \( \mathbb{F}^n \). Assume we do not know the minimum distance of \( C \). Let \( d_0 \) denote an upper bound for the minimum distance.

   **Output:** A good estimate for the minimum distance.

2. \( d_1 = d_0 \)

3. Let \( \{c\} \) denote a “small” set of random codewords in \( C \). Let \( \{v\} \) denote a corresponding set of received words, with each corresponding vector having \( [(d_1 - 1)/2] \) errors from the corresponding codeword. Compute the polynomials \( v(x) \) associated to the received words \( v \).

4. Apply algorithm 3 to try to correct the errors in the \( v \)’s. If any failure occurs, replace \( d_1 \) by \( d_1 - 1 \) and go to step 2. If all are correct, then stop and return \( d_1 \).

If this works, it’s a polynomial time algorithm for estimating the minimum distance of a cyclic code.

**References**


