From the Euclidean Algorithm for Solving a Key Equation for Dual Reed–Solomon Codes to the Berlekamp–Massey Algorithm

Conference Paper · June 2009

DOI: 10.1007/978-3-642-02181-7_4 · Source: DBLP

Michael E. O'Sullivan
San Diego State University

From the Euclidean Algorithm for Solving a Key Equation for Dual Reed–Solomon Codes to the Berlekamp-Massey Algorithm

Maria Bras-Amorós, Michael E. O'Sullivan

The Claude Shannon Institute Workshop on Coding and Cryptography

Cork, May 18, 2009

Contents

Bézout's Theorem

Given $a, b \in \mathbb{F}_q[x]$ there exist $f, g \in \mathbb{F}_q[x]$ such that

$$fa + gb = \gcd(a, b)$$

Extended Euclidean Algorithm

Let $r_{-2} = a$, $r_{-1} = b$ and, for $i \geqslant 0$ let the Euclidean division of r_{i-2} by r_{i-1} be

$$r_{i-2} = q_i r_{i-1} + r_i.$$

Define $f_{-2} = 1$, $g_{-2} = 0$, $f_{-1} = 0$, $g_{-1} = 1$, and for $i \geqslant 0$

$$f_i = f_{i-2} - q_i f_{i-1} \quad g_i = g_{i-2} - q_i g_{i-1}$$

then for all $i \geqslant 0$

$$f_i a + g_i b = r_i$$

Extended Euclidean algorithm

The extended Euclidean algorithm can be expressed in matrix form as

ALGORITHM:

Initialize:

$$\begin{pmatrix} r_{-1} & f_{-1} & g_{-1} \\ r_{-2} & f_{-2} & g_{-2} \end{pmatrix} = \begin{pmatrix} b & 0 & 1 \\ a & 1 & 0 \end{pmatrix}$$

while $\deg(r_i) \geqslant 0$**:**

$$q_i = Quotient(r_{i-2}, r_{i-1})$$

$$\begin{pmatrix} r_i & f_i & g_i \\ r_{i-1} & f_{i-1} & g_{i-1} \end{pmatrix} = \begin{pmatrix} -q_i & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} r_{i-1} & f_{i-1} & g_{i-1} \\ r_{i-2} & f_{i-2} & g_{i-2} \end{pmatrix}$$

end while

Return r_{i-1}

Extended Euclidean algorithm

Remark

1. $\deg(f_i) > \deg(f_{i-1})$ *while* $\deg(r_i) < \deg(r_{i-1})$
   $\deg(g_i) > \deg(g_{i-1})$
   (except maybe in the initial steps);
2. $\det\begin{pmatrix} r_i & f_i \\ r_{i-1} & f_{i-1} \end{pmatrix} = (-1)^{i+1}\det\begin{pmatrix} r_{-1} & f_{-1} \\ r_{-2} & f_{-2} \end{pmatrix} = (-1)^{i+1}b$, *so*

   $$f_i r_{i-1} = (-1)^i b + f_{i-1} r_i,$$

   and, since $\deg r_i < \deg r_{i-1}$ *and* $\deg f_i > \deg f_{i-1}$, *then*

   - $LT(f_i) = (-1)^i LT(b)/LT(r_{i-1})$
   - $\deg f_i = \deg b - \deg r_{i-1}$
3. $\det\begin{pmatrix} f_i & g_i \\ f_{i-1} & g_{i-1} \end{pmatrix} = (-1)^{i+1}\det\begin{pmatrix} f_{-1} & g_{-1} \\ f_{-2} & g_{-2} \end{pmatrix} = (-1)^i$, *so*

   $$f_i g_{i-1} - f_{i-1} g_i = (-1)^i$$

   and the intermediate Bézout coefficients are coprime at each step.

Extended Euclidean alg. with monic remainders

Definition

For all $i \geqslant -1$ define the matrices

$$\begin{pmatrix} R_i & F_i & G_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{G}_i \end{pmatrix} = \begin{pmatrix} 1/\mathrm{LC}(r_i) & 0 \\ 0 & (-1)^i \mathrm{LC}(r_i) \end{pmatrix} \begin{pmatrix} r_i & f_i & g_i \\ r_{i-1} & f_{i-1} & g_{i-1} \end{pmatrix}$$

Lemma

For all $i \geqslant 0$,

$$\begin{pmatrix} R_i & F_i & G_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{G}_i \end{pmatrix} = \begin{pmatrix} 1/LC(\tilde{R}_{i-1} - Q_i R_{i-1}) & 0 \\ 0 & -LC(\tilde{R}_{i-1} - Q_i R_{i-1}) \end{pmatrix} \begin{pmatrix} -Q_i & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} R_{i-1} & F_{i-1} & G_{i-1} \\ \tilde{R}_{i-1} & \tilde{F}_{i-1} & \tilde{G}_{i-1} \end{pmatrix},$$

where Q_i is the quotient of $\tilde{R}_{i-1}$ by R_{i-1}.

Extended Euclidean alg. with monic remainders

The extended Euclidean algorithm for the new matrices is

ALGORITHM:

Initialize:

$$\begin{pmatrix} R_{-1} & F_{-1} & G_{-1} \\ \tilde{R}_{-1} & \tilde{F}_{-1} & \tilde{G}_{-1} \end{pmatrix} = \begin{pmatrix} \frac{1}{\boldsymbol{LC}(b)} & 0 \\ 0 & -\boldsymbol{LC}(b) \end{pmatrix} \begin{pmatrix} b & 0 & 1 \\ a & 1 & 0 \end{pmatrix}$$

while $\deg(R_i) \geqslant 0$**:**

$$Q_i = Quotient(\tilde{R}_{i-1}, R_{i-1})$$

$$\begin{pmatrix} R_i & F_i & G_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{G}_i \end{pmatrix} = \begin{pmatrix} \frac{1}{\boldsymbol{LC}(\tilde{R}_{i-1} - Q_i R_{i-1})} & 0 \\ 0 & -\boldsymbol{LC}(\tilde{R}_{i-1} - Q_i R_{i-1}) \end{pmatrix} \begin{pmatrix} -Q_i & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} R_{i-1} & F_{i-1} & G_{i-1} \\ \tilde{R}_{i-1} & \tilde{F}_{i-1} & \tilde{G}_{i-1} \end{pmatrix}$$

end while

Return R_{i-1} **(**a constant multiple of r_{i-1}**)**

Extended Euclidean alg. with monic remainders

The extended Euclidean algorithm for the new matrices is

ALGORITHM:

Initialize:

$$\begin{pmatrix} R_{-1} & F_{-1} & G_{-1} \\ \tilde{R}_{-1} & \tilde{F}_{-1} & \tilde{G}_{-1} \end{pmatrix} = \begin{pmatrix} \frac{1}{\boldsymbol{LC}(b)} & 0 \\ 0 & -\boldsymbol{LC}(b) \end{pmatrix} \begin{pmatrix} b & 0 & 1 \\ a & 1 & 0 \end{pmatrix}$$

while $\deg(R_i) \geqslant 0$**:**

$$Q_i = Quotient(\tilde{R}_{i-1}, R_{i-1})$$

$$\begin{pmatrix} R_i & F_i & G_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{G}_i \end{pmatrix} = \begin{pmatrix} \frac{1}{\boldsymbol{LC}(\tilde{R}_{i-1} - Q_i R_{i-1})} & 0 \\ 0 & -\boldsymbol{LC}(\tilde{R}_{i-1} - Q_i R_{i-1}) \end{pmatrix} \begin{pmatrix} -Q_i & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} R_{i-1} & F_{i-1} & G_{i-1} \\ \tilde{R}_{i-1} & \tilde{F}_{i-1} & \tilde{G}_{i-1} \end{pmatrix}$$

end while

Return R_{i-1} **(**a constant multiple of r_{i-1}**)**

If $Q_i = Q_i^{(0)} + Q_i^{(1)}x + \cdots + Q_i^{(l)}x^l$ then

$$\begin{pmatrix} -Q_i & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & -Q_i^{(0)} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -Q_i^{(1)}x \\ 0 & 1 \end{pmatrix} \cdots \begin{pmatrix} 1 & -Q_i^{(l)}x^l \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

Extended Euclidean alg. with monic remainders

The extended Euclidean algorithm for the new matrices is

ALGORITHM:

Initialize:

$$\begin{pmatrix} R_{-1} & F_{-1} & G_{-1} \\ \tilde{R}_{-1} & \tilde{F}_{-1} & \tilde{G}_{-1} \end{pmatrix} = \begin{pmatrix} \frac{1}{\boldsymbol{LC}(b)} & 0 \\ 0 & -\boldsymbol{LC}(b) \end{pmatrix} \begin{pmatrix} b & 0 & 1 \\ a & 1 & 0 \end{pmatrix}$$

while $\deg(R_i) \geqslant 0$**:**

$$Q_i = Quotient(\tilde{R}_{i-1}, R_{i-1})$$

$$\begin{pmatrix} R_i & F_i & G_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{G}_i \end{pmatrix} = \begin{pmatrix} \frac{1}{\boldsymbol{LC}(\tilde{R}_{i-1} - Q_i R_{i-1})} & 0 \\ 0 & -\boldsymbol{LC}(\tilde{R}_{i-1} - Q_i R_{i-1}) \end{pmatrix} \begin{pmatrix} 1 & -Q_i^{(0)} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -Q_i^{(1)}x \\ 0 & 1 \end{pmatrix} \cdots$$
$$\cdots \begin{pmatrix} 1 & -Q_i^{(l)}x^l \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} R_{i-1} & F_{i-1} & G_{i-1} \\ \tilde{R}_{i-1} & \tilde{F}_{i-1} & \tilde{G}_{i-1} \end{pmatrix}$$

end while

Return R_{i-1} **(**a constant multiple of r_{i-1}**)**

If $Q_i = Q_i^{(0)} + Q_i^{(1)}x + \cdots + Q_i^{(l)}x^l$ then

$$\begin{pmatrix} -Q_i & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & -Q_i^{(0)} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -Q_i^{(1)}x \\ 0 & 1 \end{pmatrix} \cdots \begin{pmatrix} 1 & -Q_i^{(l)}x^l \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

Extended Euclidean alg. with monic remainders

The extended Euclidean algorithm for the new matrices is

ALGORITHM:

Initialize:

$$\begin{pmatrix} R_{-1} & F_{-1} & G_{-1} \\ \tilde{R}_{-1} & \tilde{F}_{-1} & \tilde{G}_{-1} \end{pmatrix} = \begin{pmatrix} \frac{1}{\mathbf{LC}(b)} & 0 \\ 0 & -\mathbf{LC}(b) \end{pmatrix} \begin{pmatrix} b & 0 & 1 \\ a & 1 & 0 \end{pmatrix}$$

while $\deg(R_i) \geqslant 0$**:**

$$Q_i = Quotient(\tilde{R}_{i-1}, R_{i-1})$$

$$\begin{pmatrix} R_i & F_i & G_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{G}_i \end{pmatrix} = \begin{pmatrix} \frac{1}{\mathbf{LC}(\tilde{R}_{i-1} - Q_i R_{i-1})} & 0 \\ 0 & -\mathbf{LC}(\tilde{R}_{i-1} - Q_i R_{i-1}) \end{pmatrix} \begin{pmatrix} 1 & -Q_i^{(0)} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -Q_i^{(1)}x \\ 0 & 1 \end{pmatrix} \cdots$$
$$\cdots \begin{pmatrix} 1 & -Q_i^{(l)}x^l \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} R_{i-1} & F_{i-1} & G_{i-1} \\ \tilde{R}_{i-1} & \tilde{F}_{i-1} & \tilde{G}_{i-1} \end{pmatrix}$$

end while

Return R_{i-1} **(**a constant multiple of r_{i-1}**)**

$$\left.\begin{array}{l} \mathrm{LC}(b) \\ \mathrm{LC}(\tilde{R}_{i-1} - Q_i R_{i-1}) \\ Q_i^{(j)} \end{array}\right\}$$ they all are the LC of the left-most, top-most element in the previous matrix

Extended Euclidean alg. with monic remainders

Splitting the matrix multiplications we get
ALGORITHM:

Initialize:

$$\begin{pmatrix} R_{-1} & F_{-1} & G_{-1} \\ \tilde{R}_{-1} & \tilde{F}_{-1} & \tilde{G}_{-1} \end{pmatrix} = \begin{pmatrix} b/\boldsymbol{LC}(b) & 0 & 1/\boldsymbol{LC}(b) \\ -\boldsymbol{LC}(b)a & -\boldsymbol{LC}(b) & 0 \end{pmatrix}$$

while $\deg(R_i) \geqslant 0$**:**

$$\begin{pmatrix} R_{i+1} & F_{i+1} & G_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{G}_{i+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} R_i & F_i & G_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{G}_i \end{pmatrix}$$

while $\deg(R_i) \geqslant \deg(\tilde{R}_i)$***:***

$$\begin{pmatrix} R_{i+1} & F_{i+1} & G_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{G}_{i+1} \end{pmatrix} = \begin{pmatrix} 1 & -\boldsymbol{LC}(R_i)x^{(\deg(R_i)-\deg(\tilde{R}_i))} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} R_i & F_i & G_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{G}_i \end{pmatrix}$$

end while

$$\begin{pmatrix} R_{i+1} & F_{i+1} & G_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{G}_{i+1} \end{pmatrix} = \begin{pmatrix} 1/\boldsymbol{LC}(R_i) & 0 \\ 0 & -\boldsymbol{LC}(R_i) \end{pmatrix} \begin{pmatrix} R_i & F_i & G_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{G}_i \end{pmatrix}$$

end while

Extended Euclidean alg. with monic remainders

We introduced a bunch of intermediate matrices

$$\begin{pmatrix} R_i & F_i & G_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{G}_i \end{pmatrix}$$

Not all of them satisfy

$$\begin{pmatrix} R_i & F_i & G_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{G}_i \end{pmatrix} = \begin{pmatrix} 1/\mathrm{LC}(r_i) & 0 \\ 0 & (-1)^i \mathrm{LC}(r_i) \end{pmatrix} \begin{pmatrix} r_i & f_i & g_i \\ r_{i-1} & f_{i-1} & g_{i-1} \end{pmatrix}$$

But all of them satisfy

$$F_i a + G_i b = R_i,$$
$$\tilde{F}_i a + \tilde{G}_i b = \tilde{R}_i.$$

Extended Euclidean alg. with monic remainders

The same algorithm can be expressed as

ALGORITHM:

Initialize:

$$\begin{pmatrix} R_{-1} & F_{-1} & G_{-1} \\ \tilde{R}_{-1} & \tilde{F}_{-1} & \tilde{G}_{-1} \end{pmatrix} = \begin{pmatrix} -\boldsymbol{LC}(b)a & -\boldsymbol{LC}(b) & 0 \\ b/\boldsymbol{LC}(b) & 0 & 1/\boldsymbol{LC}(b) \end{pmatrix}$$

while $\deg(R_i) \geqslant 0$**:**

$\mu = \boldsymbol{LC}(R_i)$
$p = \deg(R_i) - \deg(\tilde{R}_i)$

if $p \geqslant 0$ ***or*** $\mu = 0$ ***then***

$$\begin{pmatrix} R_{i+1} & F_{i+1} & G_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{G}_{i+1} \end{pmatrix} = \begin{pmatrix} 1 & -\mu x^p \\ 0 & 1 \end{pmatrix} \begin{pmatrix} R_i & F_i & G_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{G}_i \end{pmatrix}$$

else

$$\begin{pmatrix} R_{i+1} & F_{i+1} & G_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{G}_{i+1} \end{pmatrix} = \begin{pmatrix} x^{-p} & -\mu \\ 1/\mu & 0 \end{pmatrix} \begin{pmatrix} R_i & F_i & G_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{G}_i \end{pmatrix}$$

end if

end while

$\mu = \mathbf{LC}(R_i)$

$$\begin{pmatrix} R_{i+1} & F_{i+1} & G_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{G}_{i+1} \end{pmatrix} = \begin{pmatrix} 1/\mu & 0 \\ 0 & -\mu \end{pmatrix} \begin{pmatrix} R_i & F_i & G_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{G}_i \end{pmatrix}$$

Primal Reed-Solomon Code

Consider

- $\mathbb{F}$ a finite field of size $q = p^m$, α a primitive element in $\mathbb{F}$, $n = q - 1$.
- the identification

$$u = (u_0, u_1, \dots, u_{n-1}) \leftrightarrow u(x) = u_0 + u_1 x + \dots + u_{n-1} x^{n-1}$$

$$(\text{denote } u(\alpha) = u_0 + u_1 \alpha + \dots + u_{n-1} \alpha^{n-1})$$

The Reed-Solomon code of dimension k $C^*(k)$ is the cyclic code with generator polynomial

$$(x - \alpha)(x - \alpha^2) \cdots (x - \alpha^{n-k}).$$

It has generator and parity check matrices

$$G^*(k) = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \alpha & \alpha^2 & \dots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \dots & \alpha^{2(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha^{k-1} & \alpha^{(k-1)2} & \dots & \alpha^{(k-1)(n-1)} \end{pmatrix}, \; H^*(k) = \begin{pmatrix} 1 & \alpha & \alpha^2 & \dots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \dots & \alpha^{2(n-1)} \\ 1 & \alpha^3 & \alpha^6 & \dots & \alpha^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha^{n-k} & \alpha^{(n-k)2} & \dots & \alpha^{(n-k)(n-1)} \end{pmatrix}.$$

Primal and dual Reed-Solomon Codes

Primal Reed-Solomon Code

The Reed-Solomon code of dimension k $C^*(k)$ is the cyclic code with generator polynomial $(x-\alpha)(x-\alpha^2)\cdots(x-\alpha^{n-k})$.
It has generator and parity check matrices

$$G^*(k) = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \alpha & \alpha^2 & \dots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \dots & \alpha^{2(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha^{k-1} & \alpha^{(k-1)2} & \dots & \alpha^{(k-1)(n-1)} \end{pmatrix}, \; H^*(k) = \begin{pmatrix} 1 & \alpha & \alpha^2 & \dots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \dots & \alpha^{2(n-1)} \\ 1 & \alpha^3 & \alpha^6 & \dots & \alpha^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha^{n-k} & \alpha^{(n-k)2} & \dots & \alpha^{(n-k)(n-1)} \end{pmatrix}.$$

Dual Reed-Solomon Code

The dual Reed-Solomon code of dimension k $C(k)$ is the cyclic code with generator polynomial $(x-\alpha^{-(k+1)})\cdots(x-\alpha^{-(n-1)})(x-1)$.
It has generator and parity check matrices

$$G(k) = \begin{pmatrix} 1 & \alpha & \alpha^2 & \dots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \dots & \alpha^{2(n-1)} \\ 1 & \alpha^3 & \alpha^6 & \dots & \alpha^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha^{k} & \alpha^{2k} & \dots & \alpha^{k(n-1)} \end{pmatrix}, \; H(k) = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \alpha & \alpha^2 & \dots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \dots & \alpha^{2(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha^{n-k-1} & \alpha^{(n-k-1)2} & \dots & \alpha^{(n-k-1)(n-1)} \end{pmatrix}.$$

Primal and dual Reed-Solomon Codes

Properties

- Both codes have minimum distance $d = n - k + 1$
- $C(k)^\perp = C^*(n-k)$
- If i is a vector of dimension k and

$$\begin{aligned} c &= (c_0, c_1, \ldots, c_{n-1}) = iG(k) \in C(k), \\ c^* &= (c_0^*, c_1^*, \ldots, c_{n-1}^*) = iG^*(k) \in C^*(k), \end{aligned}$$

then

$$\begin{array}{rccccccc} c & = & (c_0^*, & \alpha c_1^*, & \alpha^2 c_2^*, & \ldots & \alpha^{n-1} c_{n-1}^*), \\ & & & & & & \\ c(\alpha^i) & = & c_0^* + & \alpha c_1^* \alpha^i + & \alpha^2 c_2^* \alpha^{2i} + & \ldots & + \alpha^{n-1} c_{n-1}^* \alpha^{(n-1)i} \\ & = & c_0^* + & c_1^* \alpha^{i+1} + & c_2^* \alpha^{2(i+1)} + & \ldots & + c_{n-1}^* \alpha^{(n-1)(i+1)} \end{array}$$

$$c(\alpha^i) = c^*(\alpha^{i+1})$$

Correction of RS codes: key polynomials

Suppose $c^* \in C^*(k)$ is the transmitted word, e^* is the error added to c^* with $t = |e^*| \leqslant \frac{d-1}{2}$, and $u^* = c^* + e^*$ is the received word.

Correction of RS codes

Error locator polynomial
$\Lambda^* = \prod_{e_i^* \neq 0}(1 - \alpha^i x)$
Error evaluator polynomial
$\Omega^* = \sum_{e_i^* \neq 0} e_i^* \alpha^i \prod_{e_j^* \neq 0, j \neq i}(1 - \alpha^i x)$

Correction of RS codes: key polynomials

Suppose $c^* \in C^*(k)$ is the transmitted word, e^* is the error added to c^* with $t = |e^*| \leqslant \frac{d-1}{2}$, and $u^* = c^* + e^*$ is the received word.

Correction of RS codes

Error locator polynomial
$\Lambda^* = \prod_{e_i^* \neq 0}(1 - \alpha^i x)$
Error evaluator polynomial
$\Omega^* = \sum_{e_i^* \neq 0} e_i^* \alpha^i \prod_{e_j^* \neq 0, j \neq i}(1 - \alpha^i x)$

Error location
$\Lambda^*(\alpha^{-i}) = 0$
Error evaluation (Forney)
$e_i^* = -\frac{\Omega^*(\alpha^{-i})}{\Lambda'^*(\alpha^{-i})}.$

Correction of RS codes: key polynomials

Suppose $c^* \in C^*(k)$ is the transmitted word, e^* is the error added to c^* with $t = |e^*| \leqslant \frac{d-1}{2}$, and $u^* = c^* + e^*$ is the received word.

Correction of RS codes

Error locator polynomial
$\Lambda^* = \prod_{e_i^* \neq 0}(1 - \alpha^i x)$
Error evaluator polynomial
$\Omega^* = \sum_{e_i^* \neq 0} e_i^* \alpha^i \prod_{e_j^* \neq 0, j \neq i}(1 - \alpha^i x)$

Error location
$\Lambda^*(\alpha^{-i}) = 0$
Error evaluation (Forney)
$e_i^* = -\frac{\Omega^*(\alpha^{-i})}{\Lambda'^*(\alpha^{-i})}.$

Syndrome polynomial
$S^* = e^*(\alpha) + e^*(\alpha^2)x + \cdots + e^*(\alpha^n)x^{n-1}$
Truncated syndrome polynomial
$\bar{S}^* = e^*(\alpha) + \cdots + e^*(\alpha^{n-k})x^{n-k-1}$

Correction of RS codes: key equations

Key equation

$\Lambda^* S^* = (1 - x^n)\Omega^*$

Truncated key equation (Berlekamp)

$\Lambda^* \bar{S}^* = \Omega^* \mod x^{n-k}$

Correction of RS codes: key equations

Key equation

$\Lambda^* S^* = (1 - x^n)\Omega^*$

Truncated key equation (Berlekamp)

$\Lambda^* \bar{S}^* = \Omega^* \mod x^{n-k}$

Bézout-like presentation

$$\underbrace{\Lambda^*}_{f_i} \underbrace{\bar{S}^*}_{\substack{a \\ \text{(known)}}} + \underbrace{m(x)}_{g_i} \underbrace{x^{n-k}}_{\substack{b \\ \text{(known)}}} = \underbrace{\Omega^*}_{r_i}$$

Correction of RS codes: key equations

Key equation

$\Lambda^* S^* = (1 - x^n)\Omega^*$

Truncated key equation (Berlekamp)

$\Lambda^* \bar{S}^* = \Omega^* \mod x^{n-k}$

Bézout-like presentation

$$\underbrace{\Lambda^*}_{f_i} \; \underbrace{\bar{S}^*}_{\substack{a \\ \text{(known)}}} + \underbrace{m(x)}_{g_i} \; \underbrace{x^{n-k}}_{\substack{b \\ \text{(known)}}} = \underbrace{\Omega^*}_{r_i}$$

Sugiyama et al's algorithm solves this by means of the ext. Euclidean algorithm.

The bound on the degree of Ω^* states the end of the algorithm.

Coprimality of Λ^* and Ω^* guarantees unicity.

Correction of RS codes: key polynomials

Suppose $c \in C(k)$ is the transmitted word, e is the error added to c with $t = |e| \leqslant \frac{d-1}{2}$, and $u = c + e$ is the received word.

Correction of RS codes

Correction of DUAL RS codes

Error locator polynomial
$\Lambda^* = \prod_{e_i^* \neq 0}(1 - \alpha^i x)$
Error evaluator polynomial
$\Omega^* = \sum_{e_i^* \neq 0} e_i^* \alpha^i \prod_{e_j^* \neq 0, j \neq i}(1 - \alpha^i x)$

Error location
$\Lambda^*(\alpha^{-i}) = 0$
Error evaluation (Forney)
$e_i^* = -\frac{\Omega^*(\alpha^{-i})}{\Lambda'^*(\alpha^{-i})}$.

Syndrome polynomial
$S^* = e^*(\alpha) + e^*(\alpha^2)x + \cdots + e^*(\alpha^n)x^{n-1}$
Truncated syndrome polynomial
$\bar{S}^* = e^*(\alpha) + \cdots + e^*(\alpha^{n-k})x^{n-k-1}$

Correction of RS codes: key polynomials

Suppose $c \in C(k)$ is the transmitted word, e is the error added to c with $t = |e| \leqslant \frac{d-1}{2}$, and $u = c + e$ is the received word.

Correction of RS codes

Error locator polynomial
$\Lambda^* = \prod_{e_i^* \neq 0}(1 - \alpha^i x)$
Error evaluator polynomial
$\Omega^* = \sum_{e_i^* \neq 0} e_i^* \alpha^i \prod_{e_j^* \neq 0, j \neq i}(1 - \alpha^i x)$

Error location
$\Lambda^*(\alpha^{-i}) = 0$
Error evaluation (Forney)
$e_i^* = -\frac{\Omega^*(\alpha^{-i})}{\Lambda'^*(\alpha^{-i})}$.

Syndrome polynomial
$S^* = e^*(\alpha) + e^*(\alpha^2)x + \cdots + e^*(\alpha^n)x^{n-1}$
Truncated syndrome polynomial
$\bar{S}^* = e^*(\alpha) + \cdots + e^*(\alpha^{n-k})x^{n-k-1}$

Correction of DUAL RS codes

Error locator polynomial
$\Lambda = \prod_{e_i \neq 0}(x - \alpha^i)$
Error evaluator polynomial
$\Omega = \sum_{e_i \neq 0} e_i \prod_{e_j \neq 0, j \neq i}(x - \alpha^i)$

Correction of RS codes: key polynomials

Suppose $c \in C(k)$ is the transmitted word, e is the error added to c with $t = |e| \leqslant \frac{d-1}{2}$, and $u = c + e$ is the received word.

Correction of RS codes

Error locator polynomial
$\Lambda^* = \prod_{e_i^* \neq 0}(1 - \alpha^i x)$
Error evaluator polynomial
$\Omega^* = \sum_{e_i^* \neq 0} e_i^* \alpha^i \prod_{e_j^* \neq 0, j \neq i}(1 - \alpha^j x)$

Error location
$\Lambda^*(\alpha^{-i}) = 0$
Error evaluation (Forney)
$e_i^* = -\frac{\Omega^*(\alpha^{-i})}{\Lambda'^*(\alpha^{-i})}.$

Syndrome polynomial
$S^* = e^*(\alpha) + e^*(\alpha^2)x + \cdots + e^*(\alpha^n)x^{n-1}$
Truncated syndrome polynomial
$\bar{S}^* = e^*(\alpha) + \cdots + e^*(\alpha^{n-k})x^{n-k-1}$

Correction of DUAL RS codes

Error locator polynomial
$\Lambda = \prod_{e_i \neq 0}(x - \alpha^i)$
Error evaluator polynomial
$\Omega = \sum_{e_i \neq 0} e_i \prod_{e_j \neq 0, j \neq i}(x - \alpha^j)$

Error location
$\Lambda(\alpha^i) = 0$
Error evaluation (Forney)
$e_i = \frac{\Omega(\alpha^i)}{\Lambda'(\alpha^i)}$

Correction of RS codes: key polynomials

Suppose $c \in C(k)$ is the transmitted word, e is the error added to c with $t = |e| \leqslant \frac{d-1}{2}$, and $u = c + e$ is the received word.

Correction of RS codes

Error locator polynomial
$\Lambda^* = \prod_{e_i^* \neq 0}(1 - \alpha^i x)$
Error evaluator polynomial
$\Omega^* = \sum_{e_i^* \neq 0} e_i^* \alpha^i \prod_{e_j^* \neq 0, j \neq i}(1 - \alpha^j x)$

Error location
$\Lambda^*(\alpha^{-i}) = 0$
Error evaluation (Forney)
$e_i^* = -\frac{\Omega^*(\alpha^{-i})}{\Lambda'^*(\alpha^{-i})}.$

Syndrome polynomial
$S^* = e^*(\alpha) + e^*(\alpha^2)x + \cdots + e^*(\alpha^n)x^{n-1}$
Truncated syndrome polynomial
$\bar{S}^* = e^*(\alpha) + \cdots + e^*(\alpha^{n-k})x^{n-k-1}$

Correction of DUAL RS codes

Error locator polynomial
$\Lambda = \prod_{e_i \neq 0}(x - \alpha^i)$
Error evaluator polynomial
$\Omega = \sum_{e_i \neq 0} e_i \prod_{e_j \neq 0, j \neq i}(x - \alpha^j)$

Error location
$\Lambda(\alpha^i) = 0$
Error evaluation (Forney)
$e_i = \frac{\Omega(\alpha^i)}{\Lambda'(\alpha^i)}$

Syndrome polynomial
$S = e(\alpha^{n-1}) + e(\alpha^{n-2})x + \cdots + e(1)x^{n-1}$
Truncated syndrome polynomial
$\bar{S} = e(\alpha^{n-k-1})x^k + \cdots + e(1)x^{n-1}.$

Correction of RS codes: key polynomials

If e and e^* are such that $e(\alpha^i) = e^*(\alpha^{i+1})$ then

$$\begin{aligned}\Lambda &= x^t \Lambda^*(1/x)\\ \Omega &= x^{t-1}\Omega^*(1/x)\end{aligned}$$

$$\begin{aligned}S &= x^{n-1}S^*(1/x)\\ \bar{S} &= x^{n-1}\bar{S}^*(1/x).\end{aligned}$$

Correction of RS codes: key equations

Key equation
$\Lambda^* S^* = (1 - x^n)\Omega^*$
Truncated key equation (Berlekamp)
$\Lambda^* \bar{S}^* = \Omega^* \mod x^{n-k}$

Key equation
$\Lambda S = (x^n - 1)\Omega$
Truncated key equation
$\deg(\Lambda \bar{S} - (x^n - 1)\Omega) < n - d/2$

Bézout-like presentation

$$\underbrace{\Lambda^*}_{f_i} \; \underbrace{\bar{S}^*}_{\substack{a \\ \text{(known)}}} + \underbrace{m(x)}_{g_i} \; \underbrace{x^{n-k}}_{\substack{b \\ \text{(known)}}} = \underbrace{\Omega^*}_{r_i}$$

Sugiyama et al's algorithm solves this
by means of the ext. Euclidean algorithm.

The bound on the degree of Ω^*
states the end of the algorithm.

Coprimality of Λ^* and Ω^*
guarantees unicity.

Correction of RS codes: key equations

Key equation
$\Lambda^* S^* = (1 - x^n)\Omega^*$
Truncated key equation (Berlekamp)
$\Lambda^* \bar{S}^* = \Omega^* \mod x^{n-k}$

Key equation
$\Lambda S = (x^n - 1)\Omega$
Truncated key equation
$\deg(\Lambda \bar{S} - (x^n - 1)\Omega) < n - d/2$

Bézout-like presentation

$$\underbrace{\Lambda^*}_{f_i} \underbrace{\bar{S}^*}_{\substack{a \\ \text{(known)}}} + \underbrace{m(x)}_{g_i} \underbrace{x^{n-k}}_{\substack{b \\ \text{(known)}}} = \underbrace{\Omega^*}_{r_i}$$

Bézout-like presentation

$$\underbrace{\Lambda}_{f_i} \underbrace{\bar{S}}_{\substack{a \\ \text{(known)}}} - \underbrace{\Omega}_{g_i} \underbrace{(x^n - 1)}_{\substack{-b \\ \text{(known)}}} = \underbrace{m(x)}_{r_i}$$

Sugiyama et al's algorithm solves this
by means of the ext. Euclidean algorithm.

The bound on the degree of Ω^*
states the end of the algorithm.

Coprimality of Λ^* and Ω^*
guarantees unicity.

Correction of RS codes: key equations

Key equation

$\Lambda^* S^* = (1 - x^n)\Omega^*$

Truncated key equation (Berlekamp)

$\Lambda^* \bar{S}^* = \Omega^* \mod x^{n-k}$

Bézout-like presentation

$$\underbrace{\Lambda^*}_{f_i} \underbrace{\bar{S}^*}_{\substack{a \\ \text{(known)}}} + \underbrace{m(x)}_{g_i} \underbrace{x^{n-k}}_{\substack{b \\ \text{(known)}}} = \underbrace{\Omega^*}_{r_i}$$

Sugiyama et al's algorithm solves this by means of the ext. Euclidean algorithm.

The bound on the degree of Ω^* states the end of the algorithm.

Coprimality of Λ^* and Ω^* guarantees unicity.

Key equation

$\Lambda S = (x^n - 1)\Omega$

Truncated key equation

$\deg(\Lambda \bar{S} - (x^n - 1)\Omega) < n - d/2$

Bézout-like presentation

$$\underbrace{\Lambda}_{f_i} \underbrace{\bar{S}}_{\substack{a \\ \text{(known)}}} - \underbrace{\Omega}_{g_i} \underbrace{(x^n - 1)}_{\substack{-b \\ \text{(known)}}} = \underbrace{m(x)}_{r_i}$$

Goal: solve this by means of the ext. Euclidean algorithm

The key equation itself states the end of the algorithm

Coprimality of Λ and Ω guarantees unicity.

Correction of RS codes: key equations

Lemma

Suppose that at most $\frac{d-1}{2}$ errors occurred. Then Λ and Ω are the unique polynomials λ, ω satisfying the following properties.

1. $\deg(\lambda\bar{S} - \omega(x^n - 1)) < n - d/2$
2. $\deg(\lambda) \leqslant d/2$
3. λ, ω *are coprime*
4. λ *is monic*

Extended Euclidean alg. for the dual key equation

ALGORITHM:

Initialize:

$$\begin{pmatrix} R_{-1} & F_{-1} & G_{-1} \\ \tilde{R}_{-1} & \tilde{F}_{-1} & \tilde{G}_{-1} \end{pmatrix} = \begin{pmatrix} \bar{S} & 1 & 0 \\ x^n - 1 & 0 & -1 \end{pmatrix}$$

while $\deg(R_i) \geqslant n - d/2$**:**

$\mu = \boldsymbol{LC}(R_i)$
$p = \deg(R_i) - \deg(\tilde{R}_i)$
if $p \geqslant 0$ ***or*** $\mu = 0$ ***then***

$$\begin{pmatrix} R_{i+1} & F_{i+1} & G_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{G}_{i+1} \end{pmatrix} = \begin{pmatrix} 1 & -\mu x^p \\ 0 & 1 \end{pmatrix} \begin{pmatrix} R_i & F_i & G_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{G}_i \end{pmatrix}$$

else

$$\begin{pmatrix} R_{i+1} & F_{i+1} & G_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{G}_{i+1} \end{pmatrix} = \begin{pmatrix} x^{-p} & -\mu \\ 1/\mu & 0 \end{pmatrix} \begin{pmatrix} R_i & F_i & G_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{G}_i \end{pmatrix}$$

end if

end while

Return F_i**,** G_i

Extended Euclidean alg. for the dual key equation

ALGORITHM:

Initialize:

$$\begin{pmatrix} R_{-1} & F_{-1} & G_{-1} \\ \tilde{R}_{-1} & \tilde{F}_{-1} & \tilde{G}_{-1} \end{pmatrix} = \begin{pmatrix} \bar{S} & 1 & 0 \\ x^n - 1 & 0 & -1 \end{pmatrix}$$

while $\deg(R_i) \geqslant n - d/2$**:**

$\mu = \boldsymbol{LC}(R_i)$

$p = \deg(R_i) - \deg(\tilde{R}_i)$

if $p \geqslant 0$ ***or*** $\mu = 0$ ***then***

$$\begin{pmatrix} R_{i+1} & F_{i+1} & G_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{G}_{i+1} \end{pmatrix} = \begin{pmatrix} 1 & -\mu x^p \\ 0 & 1 \end{pmatrix} \begin{pmatrix} R_i & F_i & G_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{G}_i \end{pmatrix}$$

else

$$\begin{pmatrix} R_{i+1} & F_{i+1} & G_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{G}_{i+1} \end{pmatrix} = \begin{pmatrix} x^{-p} & -\mu \\ 1/\mu & 0 \end{pmatrix} \begin{pmatrix} R_i & F_i & G_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{G}_i \end{pmatrix}$$

end if

end while

Return F_i**,** G_i

1. $\deg(F_i\bar{S} - G_i(x^n - 1)) = \deg(R_i) < n - d/2$

Extended Euclidean alg. for the dual key equation

ALGORITHM:

Initialize:

$$\begin{pmatrix} R_{-1} & F_{-1} & G_{-1} \\ \tilde{R}_{-1} & \tilde{F}_{-1} & \tilde{G}_{-1} \end{pmatrix} = \begin{pmatrix} \bar{S} & 1 & 0 \\ x^n - 1 & 0 & -1 \end{pmatrix}$$

while $\deg(R_i) \geqslant n - d/2$**:**

$\mu = \boldsymbol{LC}(R_i)$

$p = \deg(R_i) - \deg(\tilde{R}_i)$

if $p \geqslant 0$ ***or*** $\mu = 0$ ***then***

$$\begin{pmatrix} R_{i+1} & F_{i+1} & G_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{G}_{i+1} \end{pmatrix} = \begin{pmatrix} 1 & -\mu x^p \\ 0 & 1 \end{pmatrix} \begin{pmatrix} R_i & F_i & G_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{G}_i \end{pmatrix}$$

else

$$\begin{pmatrix} R_{i+1} & F_{i+1} & G_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{G}_{i+1} \end{pmatrix} = \begin{pmatrix} x^{-p} & -\mu \\ 1/\mu & 0 \end{pmatrix} \begin{pmatrix} R_i & F_i & G_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{G}_i \end{pmatrix}$$

end if

end while

Return F_i, G_i

2. $\deg(F_i) = n - \deg(R_{i-1}) \leqslant n - (n - d/2) = d/2$

Extended Euclidean alg. for the dual key equation

ALGORITHM:

Initialize:

$$\begin{pmatrix} R_{-1} & F_{-1} & G_{-1} \\ \tilde{R}_{-1} & \tilde{F}_{-1} & \tilde{G}_{-1} \end{pmatrix} = \begin{pmatrix} \bar{S} & 1 & 0 \\ x^n - 1 & 0 & -1 \end{pmatrix}$$

while $\deg(R_i) \geqslant n - d/2$**:**

$\mu = \boldsymbol{LC}(R_i)$
$p = \deg(R_i) - \deg(\tilde{R}_i)$
if $p \geqslant 0$ ***or*** $\mu = 0$ ***then***

$$\begin{pmatrix} R_{i+1} & F_{i+1} & G_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{G}_{i+1} \end{pmatrix} = \begin{pmatrix} 1 & -\mu x^p \\ 0 & 1 \end{pmatrix} \begin{pmatrix} R_i & F_i & G_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{G}_i \end{pmatrix}$$

else

$$\begin{pmatrix} R_{i+1} & F_{i+1} & G_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{G}_{i+1} \end{pmatrix} = \begin{pmatrix} x^{-p} & -\mu \\ 1/\mu & 0 \end{pmatrix} \begin{pmatrix} R_i & F_i & G_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{G}_i \end{pmatrix}$$

end if

end while

Return F_i, G_i

3. F_i, G_i coprime. Indeed, $\det \begin{pmatrix} F_i & G_i \\ \tilde{F}_i & \tilde{G}_i \end{pmatrix} = -1 \Rightarrow F_i(-\tilde{G}_i) + G_i(\tilde{F}_i) = 1$

Extended Euclidean alg. for the dual key equation

ALGORITHM:

Initialize:

$$\begin{pmatrix} R_{-1} & F_{-1} & G_{-1} \\ \tilde{R}_{-1} & \tilde{F}_{-1} & \tilde{G}_{-1} \end{pmatrix} = \begin{pmatrix} \bar{S} & 1 & 0 \\ x^n - 1 & 0 & -1 \end{pmatrix}$$

while $\deg(R_i) \geqslant n - d/2$**:**

$\mu = \boldsymbol{LC}(R_i)$
$p = \deg(R_i) - \deg(\tilde{R}_i)$
if $p \geqslant 0$ ***or*** $\mu = 0$ ***then***

$$\begin{pmatrix} R_{i+1} & F_{i+1} & G_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{G}_{i+1} \end{pmatrix} = \begin{pmatrix} 1 & -\mu x^p \\ 0 & 1 \end{pmatrix} \begin{pmatrix} R_i & F_i & G_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{G}_i \end{pmatrix}$$

else

$$\begin{pmatrix} R_{i+1} & F_{i+1} & G_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{G}_{i+1} \end{pmatrix} = \begin{pmatrix} x^{-p} & -\mu \\ 1/\mu & 0 \end{pmatrix} \begin{pmatrix} R_i & F_i & G_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{G}_i \end{pmatrix}$$

end if

end while

Return F_i**,** G_i

4. F_i is monic. Indeed, $\det \begin{pmatrix} R_i & F_i \\ \tilde{R}_i & \tilde{F}_i \end{pmatrix} = -1 \Rightarrow F_i(\tilde{R}_i) + R_i(-\tilde{F}_i) = 1$

Extended Euclidean alg. for the dual key equation

ALGORITHM:

Initialize:

$$\begin{pmatrix} R_{-1} & F_{-1} & G_{-1} \\ \tilde{R}_{-1} & \tilde{F}_{-1} & \tilde{G}_{-1} \end{pmatrix} = \begin{pmatrix} \bar{S} & 1 & 0 \\ x^n - 1 & 0 & -1 \end{pmatrix}$$

while $\deg(R_i) \geqslant n - d/2$**:**

$\mu = \boldsymbol{LC}(R_i)$
$p = \deg(R_i) - \deg(\tilde{R}_i)$
if $p \geqslant 0$ ***or*** $\mu = 0$ ***then***

$$\begin{pmatrix} R_{i+1} & F_{i+1} & G_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{G}_{i+1} \end{pmatrix} = \begin{pmatrix} 1 & -\mu x^p \\ 0 & 1 \end{pmatrix} \begin{pmatrix} R_i & F_i & G_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{G}_i \end{pmatrix}$$

else

$$\begin{pmatrix} R_{i+1} & F_{i+1} & G_{i+1} \\ \tilde{R}_{i+1} & \tilde{F}_{i+1} & \tilde{G}_{i+1} \end{pmatrix} = \begin{pmatrix} x^{-p} & -\mu \\ 1/\mu & 0 \end{pmatrix} \begin{pmatrix} R_i & F_i & G_i \\ \tilde{R}_i & \tilde{F}_i & \tilde{G}_i \end{pmatrix}$$

end if

end while

Return F_i**,** G_i

Theorem: If $t \leqslant \frac{d-1}{2}$ then the algorithm outputs Λ and Ω.

From Euclidean to Berlekamp-Massey

The only reason to keep the polynomials R_i (and $\tilde{R}_i$) is that we need to compute their leading coefficients (the μ's).

Lemma

$LC(R_i) = LC(F_i\bar{S})$

Proof.

On one hand, the remainder $R_i = F_i\bar{S} - G_i(x^n - 1) = F_i\bar{S} - x^nG_i + G_i$ has degree at most $n - 1$ for all $i \geqslant 0$. This means that all terms of x^nG_i cancel with terms of $F_i\bar{S}$ and that the leading term of R_i must be either a term of $F_i\bar{S}$ or a term of G_i or a sum of a term of $F_i\bar{S}$ and a term of G_i.

On the other hand, the algorithm only computes $\mathrm{LC}(R_i)$ while $\deg(R_i) \geqslant n - d/2$. We want to see that in this case the leading term of R_i has degree strictly larger than that of G_i. Indeed, one can check that for $i \geqslant 0$, $\deg(G_i) < \deg(F_i)$ and that all F_i's in the algorithm have degree at most $d/2$. So $\deg(G_i) < \deg(F_i) \leqslant d/2 \leqslant n - d/2 \leqslant \deg(R_i)$. □

From Euclidean to Berlekamp-Massey

ALGORITHM:

Initialize:

$$d_{-1} = \deg(\bar{S})$$
$$\tilde{d}_{-1} = n$$
$$\begin{pmatrix} F_{-1} & G_{-1} \\ \tilde{F}_{-1} & \tilde{G}_{-1} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

while $d_i \geqslant n - d/2$**:**

$$\mu = \textbf{\textit{Coefficient}}(F_i \bar{S}, d_i)$$
$$p = d_i - \tilde{d}_i$$

if $p \geqslant 0$ ***or*** $\mu = 0$ ***then***

$$\begin{pmatrix} F_{i+1} & G_{i+1} \\ \tilde{F}_{i+1} & \tilde{G}_{i+1} \end{pmatrix} = \begin{pmatrix} 1 & -\mu x^p \\ 0 & 1 \end{pmatrix} \begin{pmatrix} F_i & G_i \\ \tilde{F}_i & \tilde{G}_i \end{pmatrix}$$
$$d_{i+1} = d_i - 1$$
$$\tilde{d}_{i+1} = \tilde{d}_i$$

else

$$\begin{pmatrix} F_{i+1} & G_{i+1} \\ \tilde{F}_{i+1} & \tilde{G}_{i+1} \end{pmatrix} = \begin{pmatrix} x^{-p} & -\mu \\ 1/\mu & 0 \end{pmatrix} \begin{pmatrix} F_i & G_i \\ \tilde{F}_i & \tilde{G}_i \end{pmatrix}$$
$$d_{i+1} = \tilde{d}_i - 1$$
$$\tilde{d}_{i+1} = d_i$$

end if

end while

From Euclidean to Berlekamp-Massey

This last algorithm is the Berlekamp-Massey algorithm that solves the linear recurrence

$$\sum_{j=0}^{t} \Lambda_j e(\alpha^{i+j-1}) = 0$$

for all $i > 0$.

This recurrence is derived from $\Lambda \frac{S}{x^n - 1}$ being a polynomial and thus having no terms of negative order in its expression as a Laurent series in $1/x$, and from the equality

$$\frac{S}{x^n - 1} = \frac{1}{x} \left(e(1) + \frac{e(\alpha)}{x} + \frac{e(\alpha^2)}{x^2} + \cdots \right).$$

Moving back to primal Reed-Solomon codes

Suppose $c^* \in C^*(k)$ is the transmitted word, e^* is the error added to c^* and $u^* = c^* + e^*$ is the received word.

Then $c = (c_0^*, \alpha c_1^*, \alpha^2 c_2^*, \ldots, \alpha^{n-1} c_{n-1}^*) \in C(k)$ and $e = (e_0^*, \alpha e_1^*, \alpha^2 e_2^*, \ldots, \alpha^{n-1} e_{n-1}^*)$ has the same non-zero positions as e^*.

Let $u := c + e = (u_0^*, \alpha u_1^*, \alpha^2 u_2^*, \ldots, \alpha^{n-1} u_{n-1}^*)$. The error values e_i^* added to u_i^* can be computed from the error values e_i added to u_i by

$$e_i^* = e_i / \alpha^i.$$

Now we can use the previous algorithm with

$$\begin{aligned}
\bar{S} &= e(\alpha^{n-k-1})x^k + e(\alpha^{n-k-2})x^{k+1} + \cdots + e(1)x^{n-1} \\
&= e^*(\alpha^{n-k})x^k + e^*(\alpha^{n-k-1})x^{k+1} + \cdots + e^*(\alpha)x^{n-1} \\
&= u^*(\alpha^{n-k})x^k + u^*(\alpha^{n-k-1})x^{k+1} + \cdots + u^*(\alpha)x^{n-1}.
\end{aligned}$$

Once we have the error positions, we can compute the error values as

$$e_i^* = \frac{\Omega(\alpha_i)}{\alpha^i \Lambda'(\alpha^i)}.$$

Other research directions: numerical semigroups

Definition

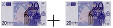
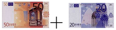
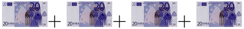
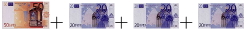
A numerical semigroup is a subset Λ of $\mathbb{N}_0$ satisfying

- $0 \in \Lambda$
- $\Lambda + \Lambda \subseteq \Lambda$
- $|\mathbb{N}_0 \setminus \Lambda|$ is finite (genus$:= g := |\mathbb{N}_0 \setminus \Lambda|$)

Example

The amounts of money one can obtain from a cash point (divided by 10)

Cash point

amount		amount/10
0		0
10	*impossible!*	
20		2
30	*impossible!*	
40		4
50		5
60		6
70		7
80		8
90		9
100		10
110		11
⋮	⋮	⋮

Let n_g denote the number of numerical semigroups of genus g.

Let n_g denote the number of numerical semigroups of genus g.

- $n_0 = 1$, since the unique numerical semigroup of genus 0 is $\mathbb{N}_0$

Let n_g denote the number of numerical semigroups of genus g.

- $n_0 = 1$, since the unique numerical semigroup of genus 0 is $\mathbb{N}_0$
- $n_1 = 1$, since the unique numerical semigroup of genus 1 is $\mathbb{N}_0 \setminus \{1\}$

Let n_g denote the number of numerical semigroups of genus g.

- $n_0 = 1$, since the unique numerical semigroup of genus 0 is $\mathbb{N}_0$
- $n_1 = 1$, since the unique numerical semigroup of genus 1 is $\mathbb{N}_0 \setminus \{1\}$
- $n_2 = 2$. Indeed the unique numerical semigroups of genus 2 are

$$\{0, 3, 4, 5, \dots\},$$

$$\{0, 2, 4, 5, \dots\}.$$

Conjecture

1. $n_g \geqslant n_{g-1} + n_{g-2}$
2. $lim_{g\to\infty} \frac{n_{g-1}+n_{g-2}}{n_g} = 1$
3. $lim_{g\to\infty} \frac{n_g}{n_{g-1}} = \phi$

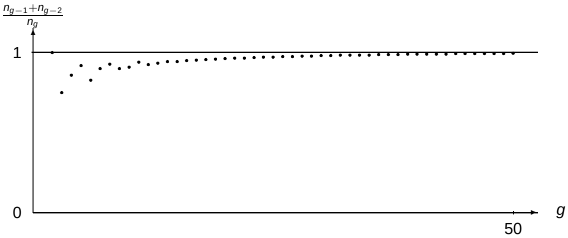
At the moment it has not even been proved that n_g is increasing.

Conjecture $n_g/n_{g-1} \longrightarrow \phi$

g	n_g	$n_{g-1}+n_{g-2}$	$\frac{n_{g-1}+n_{g-2}}{n_g}$	$\frac{n_g}{n_{g-1}}$
0	1			
1	1			1
2	2	2	1	2
3	4	3	0.75	2
4	7	6	0.857143	1.75
5	12	11	0.916667	1.71429
6	23	19	0.826087	1.91667
7	39	35	0.897436	1.69565
8	67	62	0.925373	1.71795
9	118	106	0.898305	1.76119
10	204	185	0.906863	1.72881
11	343	322	0.938776	1.68137
12	592	547	0.923986	1.72595
13	1001	935	0.934066	1.69088
14	1693	1593	0.940933	1.69131
15	2857	2694	0.942947	1.68754
16	4806	4550	0.946733	1.68218
17	8045	7663	0.952517	1.67395
18	13467	12851	0.954259	1.67396
19	22464	21512	0.957621	1.66808
20	37396	35931	0.960825	1.66471
21	62194	59860	0.962472	1.66312
22	103246	99590	0.964589	1.66006
23	170963	165440	0.967695	1.65588
24	282828	274209	0.969526	1.65432
25	467224	453791	0.971249	1.65197
26	770832	750052	0.973042	1.64981
27	1270267	1238056	0.974642	1.64792
28	2091030	2041099	0.976121	1.64613
29	3437839	3361297	0.977735	1.64409
30	5646773	5528869	0.979120	1.64254
31	9266788	9084612	0.980341	1.64108
32	15195070	14913561	0.981474	1.63973
33	24896206	24461858	0.982554	1.63844
34	40761087	40091276	0.983567	1.63724
35	66687201	65657293	0.984556	1.63605
36	109032500	107448288	0.985470	1.63498
37	178158289	175719701	0.986312	1.63399
38	290939807	287190789	0.987114	1.63304
39	474851445	469098096	0.987884	1.63213
40	774614284	765791252	0.988610	1.63128
41	1262992840	1249465729	0.989290	1.63048
42	2058356522	2037607124	0.989919	1.62975
43	3353191846	3321349362	0.990504	1.62906
44	5460401576	5411548368	0.991053	1.62842
45	8888486816	8813593422	0.991574	1.62781
46	14463633648	14348888392	0.992067	1.62723
47	23527845502	23352120464	0.992531	1.62669
48	38260496374	37991479150	0.992969	1.62618
49	62200036752	61788341876	0.993381	1.62570
50	101090300128	100460533126	0.993770	1.62525

Conjecture $n_g/n_{g-1} \to \phi$

Behavior of $\frac{n_{g-1}+n_{g-2}}{n_g}$



Behavior of $\frac{n_g}{n_{g-1}}$