Reed-Solomon codes

Coding Technology

Illés Horváth

2025/10/08

Reminder: nonbinary block codes, GF(q) for q prime

q-ary symmetric channel.

General nonbinary block coding scheme.

Hamming bound, perfect codes. Singleton bound, MDS codes.

Galois fields: finite fields with q elements.

- q prime $\rightarrow$ mod q arithmetic
- $q = p^m$ prime power $\rightarrow$ different arithmetic, to be defined later

Linear nonbinary codes

A function ψ between two linear spaces is linear if for any u_1, u_2 vectors and s_1, s_2 scalars,

$$\psi(s_1 u_1 + s_2 u_2) = s_1 \psi(u_1) + s_2 \psi(u_2).$$

Linear nonbinary codes

A function ψ between two linear spaces is linear if for any u_1, u_2 vectors and s_1, s_2 scalars,

$$\psi(s_1 u_1 + s_2 u_2) = s_1 \psi(u_1) + s_2 \psi(u_2).$$

If ψ is a $\mathrm{GF}(q)^k \to \mathrm{GF}(q)^n$ linear function, then there exists a $k \times n$ matrix G over $\mathrm{GF}(q)$ such that

$$\psi(u) = uG.$$

Linear nonbinary codes

A function ψ between two linear spaces is linear if for any u_1, u_2 vectors and s_1, s_2 scalars,

$$\psi(s_1 u_1 + s_2 u_2) = s_1 \psi(u_1) + s_2 \psi(u_2).$$

If ψ is a $\mathrm{GF}(q)^k \to \mathrm{GF}(q)^n$ linear function, then there exists a $k \times n$ matrix G over $\mathrm{GF}(q)$ such that

$$\psi(u) = uG.$$

If, for some (nonbinary) error correction code, the $\psi : u \to c$ function is linear, then we call it a linear code.

Linear nonbinary codes

A function ψ between two linear spaces is linear if for any u_1, u_2 vectors and s_1, s_2 scalars,

$$\psi(s_1 u_1 + s_2 u_2) = s_1\psi(u_1) + s_2\psi(u_2).$$

If ψ is a $\mathrm{GF}(q)^k \to \mathrm{GF}(q)^n$ linear function, then there exists a $k \times n$ matrix G over $\mathrm{GF}(q)$ such that

$$\psi(u) = uG.$$

If, for some (nonbinary) error correction code, the $\psi : u \to c$ function is linear, then we call it a linear code.

The matrix G is called the generator matrix of the code.

Linear nonbinary codes

For linear nonbinary codes, we define the parity check matrix H the same as for linear binary codes. For a $C(n, k)$ linear code with generator matrix G, we call an $(n-k) \times n$ matrix H a parity-check matrix if the rows of H are linearly independent, and

$$G \cdot H^T = 0.$$

Linear nonbinary codes

For linear nonbinary codes, we define the parity check matrix H the same as for linear binary codes. For a $C(n, k)$ linear code with generator matrix G, we call an $(n-k) \times n$ matrix H a parity-check matrix if the rows of H are linearly independent, and

$$G \cdot H^T = 0.$$

Theorem

For any generator matrix G, there always exists an H parity-check matrix.

Linear nonbinary codes

For linear nonbinary codes, we define the parity check matrix H the same as for linear binary codes. For a $C(n, k)$ linear code with generator matrix G, we call an $(n-k) \times n$ matrix H a parity-check matrix if the rows of H are linearly independent, and

$$G \cdot H^T = 0.$$

Theorem

For any generator matrix G, there always exists an H parity-check matrix.

For any linear nonbinary code,

$$d_{\min} = \min_{c \neq 0} w(c).$$

Systematic linear nonbinary codes

For systematic linear codes, G and H have a nice structure.

Theorem

Assume we have a linear code with generator matrix G. The following three properties are equivalent:

- *the code is systematic;*
- *the leftmost $k \times k$ block of G is the identity matrix;*
- *the rightmost $(n-k) \times (n-k)$ block of H is the identity matrix.*

Systematic linear nonbinary codes

For systematic linear codes, G and H have a nice structure.

Theorem

Assume we have a linear code with generator matrix G. The following three properties are equivalent:

- *the code is systematic;*
- *the leftmost $k \times k$ block of G is the identity matrix;*
- *the rightmost $(n-k) \times (n-k)$ block of H is the identity matrix.*

Moreover,

$$G = [I_k | B] \qquad \Longrightarrow \qquad H = [-B^T | I_{n-k}].$$

(B is of size $k \times (n-k)$).

Tetracode

Example. The tetracode is a $C(4,2)$ ternary $(q = 3)$ code.

$$
\begin{aligned}
(0\,0) &\to (0\,0\,0\,0)\\
(0\,1) &\to (0\,1\,1\,2)\\
(0\,2) &\to (0\,2\,2\,1)\\
(1\,0) &\to (1\,0\,1\,1)\\
(1\,1) &\to (1\,1\,2\,0)\\
(1\,2) &\to (1\,2\,0\,2)\\
(2\,0) &\to (2\,0\,2\,2)\\
(2\,1) &\to (2\,1\,0\,1)\\
(2\,2) &\to (2\,2\,1\,0)
\end{aligned}
$$

Tetracode

Example. The tetracode is a $C(4,2)$ ternary $(q = 3)$ code.

$$\begin{array}{c}
(0\,0) \to (0\,0\,0\,0) \\
(0\,1) \to (0\,1\,1\,2) \\
(0\,2) \to (0\,2\,2\,1) \\
(1\,0) \to (1\,0\,1\,1) \\
(1\,1) \to (1\,1\,2\,0) \\
(1\,2) \to (1\,2\,0\,2) \\
(2\,0) \to (2\,0\,2\,2) \\
(2\,1) \to (2\,1\,0\,1) \\
(2\,2) \to (2\,2\,1\,0)
\end{array}$$

The tetracode is a linear code with generator matrix

$$G = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \end{bmatrix}$$

Syndrome decoding

Decoding is syndrome-based:

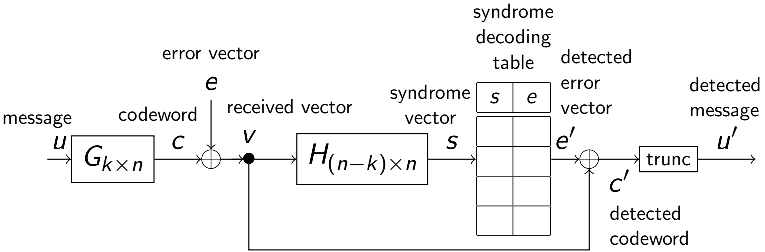
- from the received vector v, we compute the syndrome vector $s = vH^T$;
- from s, we guess the detected error vector e' using the syndrome decoding table;
- the detected codeword is $c' = v \oplus e'$;
- if the code is systematic, u' is obtained by truncation.

Syndrome decoding

Decoding is syndrome-based:

- from the received vector v, we compute the syndrome vector $s = vH^T$;
- from s, we guess the detected error vector e' using the syndrome decoding table;
- the detected codeword is $c' = v \oplus e'$;
- if the code is systematic, u' is obtained by truncation.

The general coding scheme is the following:



Reed-Solomon codes

Next we consider the $C(n, k)$ Reed-Solomon code over $GF(q)$, generated by the primitive element $\alpha \in GF(q)$.

- $1 \leq k < n \leq q - 1$

Reed-Solomon codes

Next we consider the $C(n,k)$ Reed-Solomon code over $GF(q)$, generated by the primitive element $\alpha \in GF(q)$.

- $1 \leq k < n \leq q-1$

Its generator matrix is

$$G = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \alpha & \alpha^2 & \dots & \alpha^{n-1} \\ \vdots & & & \ddots & \vdots \\ 1 & \alpha^{k-1} & \alpha^{2(k-1)} & \dots & \alpha^{(n-1)(k-1)} \end{bmatrix}.$$

Reed-Solomon codes

Next we consider the $C(n, k)$ Reed-Solomon code over $GF(q)$, generated by the primitive element $\alpha \in GF(q)$.

- $1 \leq k < n \leq q - 1$

Its generator matrix is

$$G = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \alpha & \alpha^2 & \dots & \alpha^{n-1} \\ \vdots & & & \ddots & \vdots \\ 1 & \alpha^{k-1} & \alpha^{2(k-1)} & \dots & \alpha^{(n-1)(k-1)} \end{bmatrix}.$$

(It is not systematic.)

Reed-Solomon codes

Example. The generator matrix of the C(4,2) RS code over GF(5), using the primitive element 2:

Reed-Solomon codes

Example. The generator matrix of the C(4,2) RS code over GF(5), using the primitive element 2:

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 3 \end{bmatrix}$$

Reed-Solomon codes

Example. The generator matrix of the C(4,2) RS code over GF(5), using the primitive element 2:

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 3 \end{bmatrix}$$

Compute the codewords corresponding to the message vectors $(1, 1)$ and $(3, 0)$:

Reed-Solomon codes

Example. The generator matrix of the C(4,2) RS code over GF(5), using the primitive element 2:

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 3 \end{bmatrix}$$

Compute the codewords corresponding to the message vectors $(1, 1)$ and $(3, 0)$:

$$(1\,1) \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 3 \end{bmatrix} = (2\,3\,0\,4)$$

Reed-Solomon codes

Example. The generator matrix of the C(4,2) RS code over GF(5), using the primitive element 2:

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 3 \end{bmatrix}$$

Compute the codewords corresponding to the message vectors $(1, 1)$ and $(3, 0)$:

$$(1\,1) \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 3 \end{bmatrix} = (2\,3\,0\,4)$$

$$(3\,0) \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 3 \end{bmatrix} = (3\,3\,3\,3)$$

Reed-Solomon codes

Example. The generator matrix of the C(4,2) RS code over GF(5), using the primitive element 2:

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 3 \end{bmatrix}$$

Compute the codewords corresponding to the message vectors $(1, 1)$ and $(3, 0)$:

$$(1\,1) \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 3 \end{bmatrix} = (2\,3\,0\,4)$$

$$(3\,0) \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 3 \end{bmatrix} = (3\,3\,3\,3)$$

(What is their Hamming distance?)

Reed-Solomon codes

$$G = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \alpha & \alpha^2 & \dots & \alpha^{n-1} \\ \vdots & & & \ddots & \vdots \\ 1 & \alpha^{k-1} & \alpha^{2(k-1)} & \dots & \alpha^{(n-1)(k-1)} \end{bmatrix}$$

Theorem

Reed-Solomon codes are MDS, that is,

$$d_{\min} = n - k + 1.$$

Reed-Solomon codes

$$G = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \alpha & \alpha^2 & \dots & \alpha^{n-1} \\ \vdots & & & \ddots & \vdots \\ 1 & \alpha^{k-1} & \alpha^{2(k-1)} & \dots & \alpha^{(n-1)(k-1)} \end{bmatrix}$$

Theorem

Reed-Solomon codes are MDS, that is,

$$d_{\min} = n - k + 1.$$

Proof. Let $u = (u_0 u_1 \dots u_{k-1})$ be a nonzero message vector and $c = uG = (c_0 c_1 \dots c_{n-1})$ the corresponding codeword. Then

Reed-Solomon codes

$$G = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \alpha & \alpha^2 & \dots & \alpha^{n-1} \\ \vdots & & & \ddots & \vdots \\ 1 & \alpha^{k-1} & \alpha^{2(k-1)} & \dots & \alpha^{(n-1)(k-1)} \end{bmatrix}$$

Theorem

Reed-Solomon codes are MDS, that is,

$$d_{\min} = n - k + 1.$$

Proof. Let $u = (u_0 u_1 \dots u_{k-1})$ be a nonzero message vector and $c = uG = (c_0 c_1 \dots c_{n-1})$ the corresponding codeword. Then

$$\begin{aligned} c_0 &= u_0 + u_1 + \dots + u_{k-1} \\ c_1 &= u_0 + u_1\alpha^1 + \dots + u_{k-1}\alpha^{k-1} \\ c_2 &= u_0 + u_1\alpha^2 + \dots + u_{k-1}\alpha^{2(k-1)} \\ &\vdots \\ c_{n-1} &= u_0 + u_1\alpha^{n-1} + \dots + u_{k-1}\alpha^{(n-1)(k-1)} \end{aligned}$$

Reed-Solomon codes

Using the notation

$$u(x) := u_0 + u_1 x + u_2 x^2 + \dots u_{k-1} x^{k-1},$$

this can be written as

$$\begin{aligned} c_0 &= u(1) \\ c_1 &= u(\alpha) \\ c_2 &= u(\alpha^2) \\ &\vdots \\ c_{n-1} &= u(\alpha^{n-1}) \end{aligned}$$

Reed-Solomon codes

Then

$$\begin{aligned} w(c) &= \#\{\text{nonzero coordinates of } c\} = \\ &= n - \#\{\text{zero coordinates of } c\} \geq \\ &\geq n - \#\{\text{roots of } u(x)\} \geq \\ &\geq n - \deg(u) = n - k + 1, \end{aligned}$$

Reed-Solomon codes

Then

$$\begin{aligned} w(c) = \#\{\text{nonzero coordinates of } c\} &= \\ &= n - \#\{\text{zero coordinates of } c\} \geq \\ &\geq n - \#\{\text{roots of } u(x)\} \geq \\ &\geq n - \deg(u) = n - k + 1, \end{aligned}$$

while the Singleton bound states

$$d_{\min} = \min_{c \neq 0} w(c) \leq n - k + 1,$$

so

$$d_{\min} = n - k + 1$$

and RS codes are MDS.

Reed-Solomon codes

Reed-Solomon codes are MDS:

$$d_{\text{min}} = n - k + 1,$$

so a $C(n, k)$ RS code can

- detect $n - k$ errors, and
- correct $\left\lfloor \frac{n-k}{2} \right\rfloor$ errors.

Reed-Solomon codes

Reed-Solomon codes are MDS:

$$d_{\min} = n - k + 1,$$

so a $C(n, k)$ RS code can

- detect $n - k$ errors, and
- correct $\left\lfloor \frac{n-k}{2} \right\rfloor$ errors.

RS codes are versatile: any desired error correction capability can be obtained by setting n and k accordingly.

Reed-Solomon codes

Reed-Solomon codes are MDS:

$$d_{\min} = n - k + 1,$$

so a C(n, k) RS code can

- detect $n - k$ errors, and
- correct $\left\lfloor \frac{n-k}{2} \right\rfloor$ errors.

RS codes are versatile: any desired error correction capability can be obtained by setting n and k accordingly.

Example. The C(4,2) RS code over GF(5) has

$$d_{\min} = n - k + 1 = 3$$

and it can correct $\left\lfloor \frac{n-k}{2} \right\rfloor = 1$ error.

Reed-Solomon codes

Often (but not always) n is set to the maximal possible value $n = q - 1$.

For a fixed k, this choice offers the highest error correction capability.

Reed-Solomon codes

Often (but not always) n is set to the maximal possible value $n = q - 1$.

For a fixed k, this choice offers the highest error correction capability.

RS codes with $n = q - 1$ have further nice properties (e.g. nice structure for the parity check matrix H), coming soon.

Reed-Solomon codes

Often (but not always) n is set to the maximal possible value $n = q - 1$.

For a fixed k, this choice offers the highest error correction capability.

RS codes with $n = q - 1$ have further nice properties (e.g. nice structure for the parity check matrix H), coming soon.

RS codes with $n < q - 1$ are punctured versions of the RS code with $n = q - 1$; that said, Reed-Solomon code can refer to either.

Reed-Solomon codes

Often (but not always) n is set to the maximal possible value $n = q - 1$.

For a fixed k, this choice offers the highest error correction capability.

RS codes with $n = q - 1$ have further nice properties (e.g. nice structure for the parity check matrix H), coming soon.

RS codes with $n < q - 1$ are punctured versions of the RS code with $n = q - 1$; that said, Reed-Solomon code can refer to either.

In this course, we will generally assume $n = q - 1$, but also highlight the specific differences in properties of RS codes for $n = q - 1$ and $n < q - 1$ whenever relevant.

Reed-Solomon codes

Elements of the second row of G,

$$\begin{bmatrix} 1 & \alpha & \alpha^2 & \dots & \alpha^{n-1} \end{bmatrix}$$

are called evaluation points (see the previous proof). If $n < q - 1$, sometimes different evaluation points are used. Such codes are also called Reed-Solomon codes, but we do not pursue this direction.

Reed-Solomon codes

Elements of the second row of G,

$$\begin{bmatrix} 1 & \alpha & \alpha^2 & \dots & \alpha^{n-1} \end{bmatrix}$$

are called evaluation points (see the previous proof). If $n < q - 1$, sometimes different evaluation points are used. Such codes are also called Reed-Solomon codes, but we do not pursue this direction.

RS codes are sometimes called evaluation codes for the same reason.

Reed-Solomon codes

Theorem

In case $n = q - 1$, the following H is a good parity check matrix for G:

$$H = \begin{bmatrix} 1 & \alpha & \alpha^2 & \dots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \dots & \alpha^{2(n-1)} \\ \vdots & & & \ddots & \vdots \\ 1 & \alpha^{n-k} & \alpha^{2(n-k)} & \dots & \alpha^{(n-k)(n-1)} \end{bmatrix}$$

Reed-Solomon codes

Theorem

In case $n = q - 1$, the following H is a good parity check matrix for G:

$$H = \begin{bmatrix} 1 & \alpha & \alpha^2 & \dots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \dots & \alpha^{2(n-1)} \\ \vdots & & & \ddots & \vdots \\ 1 & \alpha^{n-k} & \alpha^{2(n-k)} & \dots & \alpha^{(n-k)(n-1)} \end{bmatrix}$$

Proof. Based on G, the codeword $c = (c_0 c_1 \dots c_{n-1})$ corresponding to message $u = (u_0 u_1 \dots u_{k-1})$ has coordinates

$$c_i = \sum_{j=0}^{k-1} u_j \alpha^{ij}, \qquad 0 \leq i \leq n-1.$$

Reed-Solomon codes

Then we need to show $cH^T = 0$; coordinate ℓ of cH^T is

$$\sum_{i=0}^{n-1} c_i \alpha^{i\ell} = \sum_{i=0}^{n-1} \sum_{j=0}^{k-1} u_j \alpha^{ij} \alpha^{i\ell} = \sum_{j=0}^{k-1} u_j \sum_{i=0}^{n-1} \alpha^{i(j+\ell)}$$

Reed-Solomon codes

Then we need to show $cH^T = 0$; coordinate ℓ of cH^T is

$$\sum_{i=0}^{n-1} c_i \alpha^{i\ell} = \sum_{i=0}^{n-1} \sum_{j=0}^{k-1} u_j \alpha^{ij} \alpha^{i\ell} = \sum_{j=0}^{k-1} u_j \sum_{i=0}^{n-1} \alpha^{i(j+\ell)}$$

Here, $0 \le j \le k-1$ and $1 \le \ell \le n-k$, so $1 \le j+\ell \le n-1$, so $\alpha^{j+\ell} \ne 1$, and

$$\sum_{i=0}^{n-1} \alpha^{i(j+\ell)} = \frac{\alpha^{n(j+\ell)} - 1}{\alpha^{j+\ell} - 1} = \frac{1^{j+\ell} - 1}{\alpha^{j+\ell} - 1} = 0.$$

Reed-Solomon codes

Then we need to show $cH^T = 0$; coordinate ℓ of cH^T is

$$\sum_{i=0}^{n-1} c_i \alpha^{i\ell} = \sum_{i=0}^{n-1} \sum_{j=0}^{k-1} u_j \alpha^{ij} \alpha^{i\ell} = \sum_{j=0}^{k-1} u_j \sum_{i=0}^{n-1} \alpha^{i(j+\ell)}$$

Here, $0 \le j \le k-1$ and $1 \le \ell \le n-k$, so $1 \le j+\ell \le n-1$, so $\alpha^{j+\ell} \neq 1$, and

$$\sum_{i=0}^{n-1} \alpha^{i(j+\ell)} = \frac{\alpha^{n(j+\ell)} - 1}{\alpha^{j+\ell} - 1} = \frac{1^{j+\ell} - 1}{\alpha^{j+\ell} - 1} = 0.$$

This implies

$$\sum_{i=0}^{n-1} c_i \alpha^{i\ell} = 0 \qquad (\ell = 1, \dots, n-k),$$

so H is indeed a valid parity check matrix for G.

RS decoding

Syndrome-based decoding is fine for small RS codes, but the syndrome decoding table can get large. What is the size of the table as a function of q, n and k?

RS decoding

Syndrome-based decoding is fine for small RS codes, but the syndrome decoding table can get large. What is the size of the table as a function of q, n and k?

Syndrome vectors have length $n - k$, so the syndrome decoding table has q^{n-k} rows. Already for values like $q = 7$ and $n - k = 4$, the table has several thousand rows, which is not very practical.

RS decoding

Syndrome-based decoding is fine for small RS codes, but the syndrome decoding table can get large. What is the size of the table as a function of q, n and k?

Syndrome vectors have length $n - k$, so the syndrome decoding table has q^{n-k} rows. Already for values like $q = 7$ and $n - k = 4$, the table has several thousand rows, which is not very practical.

We look for more efficient decoding methods.

Error locator algorithm

For RS decoding, we are going to use the Error locator algorithm (also known as the Peterson-Gorenstein-Zierler algorithm).

Error locator algorithm

For RS decoding, we are going to use the Error locator algorithm (also known as the Peterson-Gorenstein-Zierler algorithm).

To reconstruct the error vector e, we need

- the number of errors $t = w(e)$ (we assume $t \leq \left\lfloor \frac{n-k}{2} \right\rfloor$);
- the location of the errors $0 \leq i_1 < i_2 \cdots < i_t \leq n-1$;
- the value of the errors $e_{i_1}, e_{i_2}, \ldots, e_{i_n}$.

Error locator algorithm

For RS decoding, we are going to use the Error locator algorithm (also known as the Peterson-Gorenstein-Zierler algorithm).

To reconstruct the error vector e, we need

- the number of errors $t = w(e)$ (we assume $t \leq \left\lfloor \frac{n-k}{2} \right\rfloor$);
- the location of the errors $0 \leq i_1 < i_2 \cdots < i_t \leq n-1$;
- the value of the errors $e_{i_1}, e_{i_2}, \ldots, e_{i_n}$.

Preparations first. We use the notation

$$X_j = \alpha^{i_j}, \qquad Y_j = e_{i_j}.$$

X_j are referred to as error locators.

Error locator algorithm

For RS decoding, we are going to use the Error locator algorithm (also known as the Peterson-Gorenstein-Zierler algorithm).

To reconstruct the error vector e, we need

- the number of errors $t = w(e)$ (we assume $t \leq \left\lfloor \frac{n-k}{2} \right\rfloor$);
- the location of the errors $0 \leq i_1 < i_2 \cdots < i_t \leq n-1$;
- the value of the errors $e_{i_1}, e_{i_2}, \ldots, e_{i_n}$.

Preparations first. We use the notation

$$X_j = \alpha^{i_j}, \qquad Y_j = e_{i_j}.$$

X_j are referred to as error locators.

The coordinates of the syndrome vector $s = eH^T = vH^T$ are $s = (s_1 \, s_2 \, \ldots \, s_{n-k})$.

Error locator algorithm

The goal is to find the e with minimal weight that satisfies $s = eH^T$.

Error locator algorithm

The goal is to find the e with minimal weight that satisfies $s = eH^T$.

The coordinates of this equation are

$$\sum_{i=0}^{n-1} e_i \alpha^{\ell_i} = \sum_{j=1}^{t} e_{i_j} \alpha^{\ell i_j} = s_\ell \qquad (\ell = 1, 2, \ldots, n-k);$$

with the X_j, Y_j notation, this is

$$\sum_{j=1}^{t} Y_j X_j^\ell = s_\ell \qquad (\ell = 1, 2, \ldots, n-k).$$

Error locator algorithm

The goal is to find the e with minimal weight that satisfies $s = eH^T$.

The coordinates of this equation are

$$\sum_{i=0}^{n-1} e_i \alpha^{\ell_i} = \sum_{j=1}^{t} e_{i_j} \alpha^{\ell i_j} = s_\ell \qquad (\ell = 1, 2, \ldots, n-k);$$

with the X_j, Y_j notation, this is

$$\sum_{j=1}^{t} Y_j X_j^\ell = s_\ell \qquad (\ell = 1, 2, \ldots, n-k).$$

This is a nonlinear system of equations with unknowns $t, X_1, \ldots, X_t, Y_1, \ldots, Y_t$.

Error locator algorithm

We aim to reduce the solution of this system to two systems of linear equations which can be solved consecutively.

The solution of the first system will provide the values of $X_1, \ldots, X_t$.

Error locator algorithm

We aim to reduce the solution of this system to two systems of linear equations which can be solved consecutively.

The solution of the first system will provide the values of $X_1, \ldots, X_t$.

Introduce the error location polynomial

$$L(x) = \prod_{i=1}^{t} (1 - xX_i) = 1 + L_1 x + \cdots + L_t x^t.$$

Error locator algorithm

We aim to reduce the solution of this system to two systems of linear equations which can be solved consecutively.

The solution of the first system will provide the values of $X_1, \ldots, X_t$.

Introduce the error location polynomial

$$L(x) = \prod_{i=1}^{t}(1 - xX_i) = 1 + L_1 x + \cdots + L_t x^t.$$

We aim to compute the polynomial $L(x)$; once $L(x)$ is known, compute its roots, which are $X_1^{-1}, \ldots, X_t^{-1}$, to get $X_1, \ldots, X_t$.

Error locator algorithm

We aim to reduce the solution of this system to two systems of linear equations which can be solved consecutively.

The solution of the first system will provide the values of $X_1, \ldots, X_t$.

Introduce the error location polynomial

$$L(x) = \prod_{i=1}^{t}(1 - xX_i) = 1 + L_1 x + \cdots + L_t x^t.$$

We aim to compute the polynomial $L(x)$; once $L(x)$ is known, compute its roots, which are $X_1^{-1}, \ldots, X_t^{-1}$, to get $X_1, \ldots, X_t$.

Once $X_1, \ldots, X_t$ are available, the equation is linear in the Y_j's, and has a unique solution.

Error locator algorithm

Since X_j^{-1} is a root of $L(x)$, for any ℓ and j we have

$$Y_j X_j^{\ell+t} L(X_j^{-1}) = 0,$$

$$\sum_{j=1}^{t} Y_j X_j^{\ell+t} L(X_j^{-1}) = 0,$$

$$\sum_{j=1}^{t} Y_j (X_j^{\ell+t} + L_1 X_j^{\ell+t-1} + \cdots + L_t X_j^{\ell}) = 0,$$

Error locator algorithm

Since X_j^{-1} is a root of $L(x)$, for any ℓ and j we have

$$Y_j X_j^{\ell+t} L(X_j^{-1}) = 0,$$

$$\sum_{j=1}^{t} Y_j X_j^{\ell+t} L(X_j^{-1}) = 0,$$

$$\sum_{j=1}^{t} Y_j (X_j^{\ell+t} + L_1 X_j^{\ell+t-1} + \cdots + L_t X_j^{\ell}) = 0,$$

which simplifies to

$$L_1 s_{\ell+t-1} + L_2 s_{\ell+t-2} + \cdots + L_t s_t = -s_{\ell+t} \qquad (\ell = 1, \ldots, t)$$

Error locator algorithm

Introducing more notation:

$$U_r = \begin{bmatrix} s_1 & s_2 & \dots & s_r \\ s_2 & s_3 & \dots & s_{r+1} \\ \vdots & \vdots & \ddots & \vdots \\ s_r & s_{r+1} & \dots & s_{2r-1} \end{bmatrix}$$

then we obtain the system of linear equations

$$\begin{bmatrix} L_t & L_{t-1} & \dots & L_1 \end{bmatrix} \cdot U_t^T = \begin{bmatrix} -s_{t+1} & -s_{t+2} & \dots & -s_{2t} \end{bmatrix}.$$

Error locator algorithm

Introducing more notation:

$$U_r = \begin{bmatrix} s_1 & s_2 & \dots & s_r \\ s_2 & s_3 & \dots & s_{r+1} \\ \vdots & \vdots & \ddots & \vdots \\ s_r & s_{r+1} & \dots & s_{2r-1} \end{bmatrix}$$

then we obtain the system of linear equations

$$\begin{bmatrix} L_t & L_{t-1} & \dots & L_1 \end{bmatrix} \cdot U_t^T = \begin{bmatrix} -s_{t+1} & -s_{t+2} & \dots & -s_{2t} \end{bmatrix}.$$

$\det U_t \neq 0$, but $\det U_r = 0$ if $r > t$ (no proof, but based on Vandermonde structure).

Error locator algorithm

Based on the above, the Error locator algorithm is the following:

1. Compute $s_1, s_2, \ldots s_{n-k}$.
2. Find the largest r for which U_r is invertible. This will give the value of t.
3. Solve

$$\begin{bmatrix} L_t & L_{t-1} & \ldots & L_1 \end{bmatrix} \cdot U_t^T = \begin{bmatrix} -s_{t+1} & -s_{t+2} & \ldots & -s_{2t} \end{bmatrix}.$$

   to obtain the $L_1, \ldots, L_t$ values.
4. Find the roots of $L(x)$, then the inverse of the roots are $X_1, X_2, \ldots, X_t$.
5. Solve the system of equations

$$\sum_{j=1}^{t} Y_j X_j^{\ell} = s_{\ell} \qquad (\ell = 1, 2, \ldots, n-k)$$

   which is linear for Y_j since the X_j's are available.
6. Compute the error vector e from the X_j's and Y_j's.

Error locator algorithm

For small $n-k$, the Error locator algorithm is computationally fast.

Error locator algorithm

For small $n - k$, the Error locator algorithm is computationally fast.

The Error locator algorithm can be used to replace syndrome decoding; it is fast enough to be used online to compute the detected error e'.

Error locator algorithm

For small $n - k$, the Error locator algorithm is computationally fast.

The Error locator algorithm can be used to replace syndrome decoding; it is fast enough to be used online to compute the detected error e'.

For larger values of $n - k$, some of the steps of the algorithm can be replaced by more efficient calculations. Steps 2 and 3 can be replaced by the Berlekamp-Massey algorithm, and Step 5 can be replaced by the Forney-algorithm. We do not pursue these results.

Error locator algorithm

For small $n-k$, the Error locator algorithm is computationally fast.

The Error locator algorithm can be used to replace syndrome decoding; it is fast enough to be used online to compute the detected error e'.

For larger values of $n-k$, some of the steps of the algorithm can be replaced by more efficient calculations. Steps 2 and 3 can be replaced by the Berlekamp-Massey algorithm, and Step 5 can be replaced by the Forney-algorithm. We do not pursue these results.

Similar versions of the algorithm can be used for other codes, not just RS.

Error locator algorithm

For small $n-k$, the Error locator algorithm is computationally fast.

The Error locator algorithm can be used to replace syndrome decoding; it is fast enough to be used online to compute the detected error e'.

For larger values of $n-k$, some of the steps of the algorithm can be replaced by more efficient calculations. Steps 2 and 3 can be replaced by the Berlekamp-Massey algorithm, and Step 5 can be replaced by the Forney-algorithm. We do not pursue these results.

Similar versions of the algorithm can be used for other codes, not just RS.

We will also discuss another decoder for RS codes - after some more preparation.

Reed-Solomon codes

Example. The codewords of the C(4,2) RS code over GF(5) using the primitive element 2:

$(0\,0) \to (0\,0\,0\,0)$	$(2\,0) \to (2\,2\,2\,2)$	$(4\,0) \to (4\,4\,4\,4)$
$(0\,1) \to (1\,2\,4\,3)$	$(2\,1) \to (3\,4\,1\,0)$	$(4\,1) \to (0\,1\,3\,2)$
$(0\,2) \to (2\,4\,3\,1)$	$(2\,2) \to (4\,1\,0\,3)$	$(4\,2) \to (1\,3\,2\,0)$
$(0\,3) \to (3\,1\,2\,4)$	$(2\,3) \to (0\,3\,4\,1)$	$(4\,3) \to (2\,0\,1\,3)$
$(0\,4) \to (4\,3\,1\,2)$	$(2\,4) \to (1\,0\,3\,4)$	$(4\,4) \to (3\,2\,0\,1)$
$(1\,0) \to (1\,1\,1\,1)$	$(3\,0) \to (3\,3\,3\,3)$	
$(1\,1) \to (2\,3\,0\,4)$	$(3\,1) \to (4\,0\,2\,1)$	
$(1\,2) \to (3\,0\,4\,2)$	$(3\,2) \to (0\,2\,1\,4)$	
$(1\,3) \to (4\,2\,3\,0)$	$(3\,3) \to (1\,4\,0\,2)$	
$(1\,4) \to (0\,4\,2\,3)$	$(3\,4) \to (2\,1\,4\,0)$	

Cyclic codes

A code is cyclic if for any codeword

$$c = (c_0\, c_1\, c_2 \dots c_{n-1}),$$

its cyclically shifted version

$$Sc = (c_{n-1}\, c_0\, c_1 \dots c_{n-2})$$

is also a codeword. S is the cyclic shift operator.

Cyclic codes

A code is cyclic if for any codeword

$$c = (c_0\, c_1\, c_2 \ldots c_{n-1}),$$

its cyclically shifted version

$$Sc = (c_{n-1}\, c_0\, c_1 \ldots c_{n-2})$$

is also a codeword. S is the cyclic shift operator.

Example. The previous C(4,2) RS code over GF(5) is cyclic.

Cyclic codes

A code is cyclic if for any codeword

$$c = (c_0\, c_1\, c_2 \ldots c_{n-1}),$$

its cyclically shifted version

$$Sc = (c_{n-1}\, c_0\, c_1 \ldots c_{n-2})$$

is also a codeword. S is the cyclic shift operator.

Example. The previous C(4,2) RS code over GF(5) is cyclic.

Motto: the more highly structured a code is, the more efficiently it can be represented and calculated – with the proper tools.

Cyclic codes

A code is cyclic if for any codeword

$$c = (c_0\, c_1\, c_2 \ldots c_{n-1}),$$

its cyclically shifted version

$$Sc = (c_{n-1}\, c_0\, c_1 \ldots c_{n-2})$$

is also a codeword. S is the cyclic shift operator.

Example. The previous C(4,2) RS code over GF(5) is cyclic.

Motto: the more highly structured a code is, the more efficiently it can be represented and calculated – with the proper tools.

Linear cyclic codes can be described very efficiently using code polynomials.

Code polynomials

So far, the language used to describe linear codes has been vectors and matrices. But there is another language to do that, with polynomials, which can be even better.

Code polynomials

So far, the language used to describe linear codes has been vectors and matrices. But there is another language to do that, with polynomials, which can be even better.

We assign polynomials over GF(q) to messages and codewords.

For any message vector $u = (u_0 u_1 \dots u_{k-1}) \in \mathrm{GF}(q)^k$,

$$u(x) = u_0 + u_1 x + \cdots + u_{k-1} x^{k-1}.$$

Code polynomials

So far, the language used to describe linear codes has been vectors and matrices. But there is another language to do that, with polynomials, which can be even better.

We assign polynomials over GF(q) to messages and codewords.

For any message vector $u = (u_0 u_1 \dots u_{k-1}) \in \mathrm{GF}(q)^k$,

$$u(x) = u_0 + u_1 x + \cdots + u_{k-1} x^{k-1}.$$

For any codeword $c = (c_0 c_1 \dots c_{n-1}) \in \mathrm{GF}(q)^n$,

$$c(x) = c_0 + c_1 x + \cdots + c_{n-1} x^{n-1}$$

is the corresponding code polynomial.

Code polynomials

So far, the language used to describe linear codes has been vectors and matrices. But there is another language to do that, with polynomials, which can be even better.

We assign polynomials over GF(q) to messages and codewords.

For any message vector $u = (u_0 u_1 \dots u_{k-1}) \in \mathrm{GF}(q)^k$,

$$u(x) = u_0 + u_1 x + \cdots + u_{k-1} x^{k-1}.$$

For any codeword $c = (c_0 c_1 \dots c_{n-1}) \in \mathrm{GF}(q)^n$,

$$c(x) = c_0 + c_1 x + \cdots + c_{n-1} x^{n-1}$$

is the corresponding code polynomial.

We can similarly assign polynomials to error vectors, received vectors etc.

Code polynomials

So far, the language used to describe linear codes has been vectors and matrices. But there is another language to do that, with polynomials, which can be even better.

We assign polynomials over GF(q) to messages and codewords.

For any message vector $u = (u_0 u_1 \dots u_{k-1}) \in \mathrm{GF}(q)^k$,

$$u(x) = u_0 + u_1 x + \cdots + u_{k-1} x^{k-1}.$$

For any codeword $c = (c_0 c_1 \dots c_{n-1}) \in \mathrm{GF}(q)^n$,

$$c(x) = c_0 + c_1 x + \cdots + c_{n-1} x^{n-1}$$

is the corresponding code polynomial.

We can similarly assign polynomials to error vectors, received vectors etc.

Note that the terms are in increasing order of degree.

Polynomials over GF(q)

Brief summary of polynomials over GF(q).

$a(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_m x^m;\ a_0, a_1, a_2, \ldots, a_m \in \mathrm{GF}(q)$

Polynomials over GF(q)

Brief summary of polynomials over GF(q).

$a(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_m x^m$; $a_0, a_1, a_2, \ldots, a_m \in \mathrm{GF}(q)$

Roots: $x_1, \ldots, x_m \in \mathrm{GF}(q)$: $a(x_i) = 0$, $i = 1, \ldots, m$

If $\deg(a(x)) = m$, then $a(x)$ has $\leq m$ roots.

If $\deg(a(x)) = m$ and $a(x)$ has m roots $x_1, \ldots, x_m$, then

$$a(x) = a_m \prod_{i=1}^{m} (x - x_i).$$

Polynomials over GF(q)

Polynomial division with remainder: given $a(x)$ and $d(x)$ with $\deg(a(x)) = m > \deg(d(x)) = k$,

$$\exists q(x), r(x) : \quad a(x) = q(x)d(x) + r(x); \quad \deg(r(x)) < k.$$

Polynomials over GF(q)

Polynomial division with remainder: given $a(x)$ and $d(x)$ with $\deg(a(x)) = m > \deg(d(x)) = k$,

$$\exists q(x), r(x): \quad a(x) = q(x)d(x) + r(x); \quad \deg(r(x)) < k.$$

We also use the notation

$$a(x) \mod d(x) = r(x).$$

Polynomials over GF(q)

Polynomial division with remainder: given $a(x)$ and $d(x)$ with $\deg(a(x)) = m > \deg(d(x)) = k$,

$$\exists q(x), r(x) : \quad a(x) = q(x)d(x) + r(x); \quad \deg(r(x)) < k.$$

We also use the notation

$$a(x) \mod d(x) = r(x).$$

Calculating polynomial division is similar to CRC coding, but in each step, we may also need to adjust the main coefficient by multiplication.

Polynomials over GF(q)

Polynomial division with remainder: given $a(x)$ and $d(x)$ with $\deg(a(x)) = m > \deg(d(x)) = k$,

$$\exists q(x), r(x): \quad a(x) = q(x)d(x) + r(x); \quad \deg(r(x)) < k.$$

We also use the notation

$$a(x) \mod d(x) = r(x).$$

Calculating polynomial division is similar to CRC coding, but in each step, we may also need to adjust the main coefficient by multiplication.

We will also discuss dedicated architectures for polynomial multiplication and division.

Polynomials over GF(q)

Polynomial division with remainder: given $a(x)$ and $d(x)$ with $\deg(a(x)) = m > \deg(d(x)) = k$,

$$\exists q(x), r(x) : \quad a(x) = q(x)d(x) + r(x); \quad \deg(r(x)) < k.$$

We also use the notation

$$a(x) \mod d(x) = r(x).$$

Calculating polynomial division is similar to CRC coding, but in each step, we may also need to adjust the main coefficient by multiplication.

We will also discuss dedicated architectures for polynomial multiplication and division.

For polynomials, the cyclic shift operator is

$$Sc(x) = xc(x) \mod x^n - 1$$

Linear cyclic codes

Theorem

For any $C(n,k)$ cyclic linear code, there is a unique $g(x)$ of degree $n-k$ with main coefficient $g_{n-k}=1$ such that for any vector c of length n,

$$c \text{ is a codeword} \iff g(x)|c(x).$$

Linear cyclic codes

Theorem

For any $C(n,k)$ cyclic linear code, there is a unique $g(x)$ of degree $n-k$ with main coefficient $g_{n-k}=1$ such that for any vector c of length n,

$$c \text{ is a codeword} \iff g(x)|c(x).$$

$g(x)$ is called the generator polynomial of the code.

Linear cyclic codes

Proof. Assume a code is linear and cyclic. From among the codewords, select the one whose code polynomial has minimal degree. Due to linearity, we can assume the main coefficient is 1 (otherwise we just divide by the main coefficient).

Linear cyclic codes

Proof. Assume a code is linear and cyclic. From among the codewords, select the one whose code polynomial has minimal degree. Due to linearity, we can assume the main coefficient is 1 (otherwise we just divide by the main coefficient).

Let this polynomial be $g(x)$, with $\deg(g(x)) = r$. We aim to prove the following:

(a) for any polynomial $u(x)$ with $\deg(u(x)) \leq n - r - 1$, $g(x)u(x)$ is a code polynomial;

(b) $g(x)$ divides every code polynomial;

(c) there is no other $g'(x)$ with these properties;

(d) $\deg(g(x)) = r = n - k$.

Linear cyclic codes

(a) The code is cyclic and $g(x)$ is a code polynomial, so

$$g(x), xg(x), x^2g(x), \dots, x^{n-r-1}g(x)$$

are all code polynomials, and due to linearity, for any $u(x) = u_0 + u_1x + \cdots + u_{n-r-1}x^{n-r-1}$,

$$u(x)g(x) = u_0g(x) + u_1xg(x) + \cdots + u_{n-r-1}x^{n-r-1}g(x)$$

is also a code polynomial.

Linear cyclic codes

(b) For any $c(x)$ code polynomial, either $g(x)|c(x)$, or the polynomial division

$$r(x) = c(x) \mod g(x)$$

has a nonzero remainder $r(x)$ with $\deg(r(x)) < \deg(g(x))$. But then $r(x)$ is also a code polynomial (due to linear and cyclic code), which contradicts $g(x)$ having minimal degree. So $g(x)|c(x)$.

Linear cyclic codes

(b) For any $c(x)$ code polynomial, either $g(x)|c(x)$, or the polynomial division

$$r(x) = c(x) \mod g(x)$$

has a nonzero remainder $r(x)$ with $\deg(r(x)) < \deg(g(x))$. But then $r(x)$ is also a code polynomial (due to linear and cyclic code), which contradicts $g(x)$ having minimal degree. So $g(x)|c(x)$.

(c) If there was another $g'(x)$ with the same properties as $g(x)$, then $g(x) - g'(x)$ would be a code polynomial that divides all code polynomials, which once again contradicts $\deg(g(x))$ being minimal.

Linear cyclic codes

(b) For any $c(x)$ code polynomial, either $g(x)|c(x)$, or the polynomial division

$$r(x) = c(x) \mod g(x)$$

has a nonzero remainder $r(x)$ with $\deg(r(x)) < \deg(g(x))$. But then $r(x)$ is also a code polynomial (due to linear and cyclic code), which contradicts $g(x)$ having minimal degree. So $g(x)|c(x)$.

(c) If there was another $g'(x)$ with the same properties as $g(x)$, then $g(x) - g'(x)$ would be a code polynomial that divides all code polynomials, which once again contradicts $\deg(g(x))$ being minimal.

(d) If $\deg(g(x)) = r$, then $g(x)$ has q^{n-r} multiples, which must be equal to the number of codewords, which is q^k for a $C(n, k)$ code, and $r = n - k$ follows.

Linear cyclic codes

Theorem

For any $C(n,k)$ linear cyclic code,

$$g(x) | x^n - 1.$$

Linear cyclic codes

Theorem

For any $C(n,k)$ linear cyclic code,

$$g(x)|x^n - 1.$$

Conversely, for any polynomial $g(x)|x^n - 1$ with $\deg(g(x)) = n - k$ and main coefficient $g_{n-k} = 1$, there is a $C(n,k)$ linear cyclic code with generator polynomial $g(x)$.

Linear cyclic codes

Theorem

For any $C(n, k)$ linear cyclic code,

$$g(x) | x^n - 1.$$

Conversely, for any polynomial $g(x) | x^n - 1$ with $\deg(g(x)) = n - k$ and main coefficient $g_{n-k} = 1$, there is a $C(n, k)$ linear cyclic code with generator polynomial $g(x)$.

Proof. Since g has degree $n - k$,

$$S^{k-1}g(x) = x^{k-1}g(x),$$
$$S^k g(x) = x^{k-1}g(x) - (x^n - 1)$$

are both code polynomials, so divisible by $g(x)$, but then

$$g(x) | S^{k-1}g(x) - S^k g(x) = x^n - 1.$$

Linear cyclic codes

For the converse, consider a polynomial $g(x) | x^n - 1$ with $\deg(g(x)) = n - k$. Then the set of codewords corresponding to multiples of $g(x)$ is...

- clearly linear, and
- has q^k elements,
- so we only need to prove that it is cyclic.

Linear cyclic codes

For the converse, consider a polynomial $g(x)|x^n - 1$ with $\deg(g(x)) = n - k$. Then the set of codewords corresponding to multiples of $g(x)$ is...

- clearly linear, and
- has q^k elements,
- so we only need to prove that it is cyclic.

Take a polynomial $c(x) = a(x)g(x)$ from this set. We need to prove $Sc(x)$ is also a multiple of $g(x)$.

- If $\deg(a(x)) < k - 1$, then $Sc(x) = xc(x) = xa(x)g(x)$.

Linear cyclic codes

For the converse, consider a polynomial $g(x)|x^n - 1$ with $\deg(g(x)) = n - k$. Then the set of codewords corresponding to multiples of $g(x)$ is...

- clearly linear, and
- has q^k elements,
- so we only need to prove that it is cyclic.

Take a polynomial $c(x) = a(x)g(x)$ from this set. We need to prove $Sc(x)$ is also a multiple of $g(x)$.

- If $\deg(a(x)) < k - 1$, then $Sc(x) = xc(x) = xa(x)g(x)$.
- If $\deg(a(x)) = k - 1$, then $Sc(x) = xc(x) - a_{k-1}(x^n - 1)$, which is also divisible by $g(x)$ due to $g(x)|x^n - 1$.

Linear cyclic codes

Basically, the previous two theorems say that there is a one-to-one correspondence between cyclic linear codes and generator polynomials.

So, are RS codes cyclic linear codes? If yes, then $g(x)$ is a code polynomial with minimal degree and main coefficient 1.

Linear cyclic codes

Basically, the previous two theorems say that there is a one-to-one correspondence between cyclic linear codes and generator polynomials.

So, are RS codes cyclic linear codes? If yes, then $g(x)$ is a code polynomial with minimal degree and main coefficient 1.

The list of code polynomials can be obtained by converting all codewords into polynomials, for example,

$$(1\,2\,4\,3) \to 1 + 2x + 4x^2 + 3x^3$$
$$(4\,2\,3\,0) \to 4 + 2x + 3x^2$$

C(4,2) RS code

Can you find the vector corresponding to the generator polynomial from the list of codewords of the C(4,2) RS code over GF(5) using the primitive element 2?

$(0\,0) \rightarrow (0\,0\,0\,0)$	$(2\,0) \rightarrow (2\,2\,2\,2)$	$(4\,0) \rightarrow (4\,4\,4\,4)$
$(0\,1) \rightarrow (1\,2\,4\,3)$	$(2\,1) \rightarrow (3\,4\,1\,0)$	$(4\,1) \rightarrow (0\,1\,3\,2)$
$(0\,2) \rightarrow (2\,4\,3\,1)$	$(2\,2) \rightarrow (4\,1\,0\,3)$	$(4\,2) \rightarrow (1\,3\,2\,0)$
$(0\,3) \rightarrow (3\,1\,2\,4)$	$(2\,3) \rightarrow (0\,3\,4\,1)$	$(4\,3) \rightarrow (2\,0\,1\,3)$
$(0\,4) \rightarrow (4\,3\,1\,2)$	$(2\,4) \rightarrow (1\,0\,3\,4)$	$(4\,4) \rightarrow (3\,2\,0\,1)$
$(1\,0) \rightarrow (1\,1\,1\,1)$	$(3\,0) \rightarrow (3\,3\,3\,3)$	
$(1\,1) \rightarrow (2\,3\,0\,4)$	$(3\,1) \rightarrow (4\,0\,2\,1)$	
$(1\,2) \rightarrow (3\,0\,4\,2)$	$(3\,2) \rightarrow (0\,2\,1\,4)$	
$(1\,3) \rightarrow (4\,2\,3\,0)$	$(3\,3) \rightarrow (1\,4\,0\,2)$	
$(1\,4) \rightarrow (0\,4\,2\,3)$	$(3\,4) \rightarrow (2\,1\,4\,0)$	

C(4,2) RS code

Can you find the vector corresponding to the generator polynomial from the list of codewords of the C(4,2) RS code over GF(5) using the primitive element 2?

$(0\,0) \to (0\,0\,0\,0)$	$(2\,0) \to (2\,2\,2\,2)$	$(4\,0) \to (4\,4\,4\,4)$
$(0\,1) \to (1\,2\,4\,3)$	$(2\,1) \to (3\,4\,1\,0)$	$(4\,1) \to (0\,1\,3\,2)$
$(0\,2) \to (2\,4\,3\,1)$	$(2\,2) \to (4\,1\,0\,3)$	$(4\,2) \to (1\,3\,2\,0)$
$(0\,3) \to (3\,1\,2\,4)$	$(2\,3) \to (0\,3\,4\,1)$	$(4\,3) \to (2\,0\,1\,3)$
$(0\,4) \to (4\,3\,1\,2)$	$(2\,4) \to (1\,0\,3\,4)$	$(4\,4) \to (3\,2\,0\,1)$
$(1\,0) \to (1\,1\,1\,1)$	$(3\,0) \to (3\,3\,3\,3)$	
$(1\,1) \to (2\,3\,0\,4)$	$(3\,1) \to (4\,0\,2\,1)$	
$(1\,2) \to (3\,0\,4\,2)$	$(3\,2) \to (0\,2\,1\,4)$	
$(1\,3) \to (4\,2\,3\,0)$	$(3\,3) \to (1\,4\,0\,2)$	
$(1\,4) \to (0\,4\,2\,3)$	$(3\,4) \to (2\,1\,4\,0)$	

It should end with as many 0's as possible, with a 1 before that

C(4,2) RS code

Can you find the vector corresponding to the generator polynomial from the list of codewords of the C(4,2) RS code over GF(5) using the primitive element 2?

$(0\,0) \to (0\,0\,0\,0)$	$(2\,0) \to (2\,2\,2\,2)$	$(4\,0) \to (4\,4\,4\,4)$
$(0\,1) \to (1\,2\,4\,3)$	$(2\,1) \to (3\,4\,1\,0)$	$(4\,1) \to (0\,1\,3\,2)$
$(0\,2) \to (2\,4\,3\,1)$	$(2\,2) \to (4\,1\,0\,3)$	$(4\,2) \to (1\,3\,2\,0)$
$(0\,3) \to (3\,1\,2\,4)$	$(2\,3) \to (0\,3\,4\,1)$	$(4\,3) \to (2\,0\,1\,3)$
$(0\,4) \to (4\,3\,1\,2)$	$(2\,4) \to (1\,0\,3\,4)$	$(4\,4) \to (3\,2\,0\,1)$
$(1\,0) \to (1\,1\,1\,1)$	$(3\,0) \to (3\,3\,3\,3)$	
$(1\,1) \to (2\,3\,0\,4)$	$(3\,1) \to (4\,0\,2\,1)$	
$(1\,2) \to (3\,0\,4\,2)$	$(3\,2) \to (0\,2\,1\,4)$	
$(1\,3) \to (4\,2\,3\,0)$	$(3\,3) \to (1\,4\,0\,2)$	
$(1\,4) \to (0\,4\,2\,3)$	$(3\,4) \to (2\,1\,4\,0)$	

It should end with as many 0's as possible, with a 1 before that $\to$ $g = (3\,4\,1\,0)$ and

$$g(x) = 3 + 4x + x^2.$$

Reed-Solomon codes

Theorem

A $C(n,k)$ RS code over $GF(q)$ with $n = q - 1$ using primitive element α is a cyclic linear code with generator polynomial

$$g(x) = \prod_{i=1}^{n-k} (x - \alpha^i).$$

Reed-Solomon codes

Theorem

A $C(n,k)$ RS code over $GF(q)$ with $n = q-1$ using primitive element α is a cyclic linear code with generator polynomial

$$g(x) = \prod_{i=1}^{n-k} (x - \alpha^i).$$

Remark. For $n < q-1$, the code is not cyclic.

Reed-Solomon codes

Theorem

A $C(n,k)$ RS code over $GF(q)$ with $n = q - 1$ using primitive element α is a cyclic linear code with generator polynomial

$$g(x) = \prod_{i=1}^{n-k} (x - \alpha^i).$$

Remark. For $n < q - 1$, the code is not cyclic.

Example. For the C(4,2) RS code over GF(5) using the primitive element 2, the generator polynomial is

$$g(x) = (x - 2)(x - 2^2) = (3 + x)(1 + x) = 3 + 4x + x^2.$$

Reed-Solomon codes

Theorem

A C(n, k) RS code over GF(q) with $n = q - 1$ using primitive element α is a cyclic linear code with generator polynomial

$$g(x) = \prod_{i=1}^{n-k} (x - \alpha^i).$$

Remark. For $n < q - 1$, the code is not cyclic.

Example. For the C(4,2) RS code over GF(5) using the primitive element 2, the generator polynomial is

$$g(x) = (x - 2)(x - 2^2) = (3 + x)(1 + x) = 3 + 4x + x^2.$$

A few code polynomials:

$$(1\,2\,4\,3) \to 1 + 2x + 4x^2 + 3x^3 = (2 + 3x)(3 + 4x + x^2),$$

$$(0\,3\,4\,1) \to 3x + 4x^2 + x^3 = x(3 + 4x + x^2),$$

$$(4\,4\,4\,4) \to 4 + 4x + 4x^2 + 4x^3 = (3 + 4x)(3 + 4x + x^2).$$

Reed-Solomon codes

Proof. The generator matrix G and parity check matrix H of the $C(n,k)$ RS code with $n = q - 1$, generated by primitive element α are

$$G = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \alpha & \alpha^2 & \dots & \alpha^{n-1} \\ \vdots & & & \ddots & \vdots \\ 1 & \alpha^{k-1} & \alpha^{2(k-1)} & \dots & \alpha^{(n-1)(k-1)} \end{bmatrix}$$

$$H = \begin{bmatrix} 1 & \alpha & \alpha^2 & \dots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \dots & \alpha^{2(n-1)} \\ \vdots & & & \ddots & \vdots \\ 1 & \alpha^{n-k} & \alpha^{2(n-k)} & \dots & \alpha^{(n-k)(n-1)} \end{bmatrix}$$

Reed-Solomon codes

For a codeword $c = (c_0 c_1 \dots c_n)$, $cH^T = 0$ means

$$\sum_{j=0}^{n-1} c_j \alpha^{ij} = 0 \qquad i = 1, \dots, n-k$$

Reed-Solomon codes

For a codeword $c = (c_0 c_1 \dots c_n)$, $cH^T = 0$ means

$$\sum_{j=0}^{n-1} c_j \alpha^{ij} = 0 \qquad i = 1, \dots, n-k$$

But this means that $\alpha^1, \dots, \alpha^{n-k}$ are roots of $c(x)$, so

$$g(x) = \prod_{i=1}^{n-k} (x - \alpha^i) \Big| c(x)$$

for every code polynomial.

Reed-Solomon codes

For a codeword $c = (c_0 c_1 \dots c_n)$, $cH^T = 0$ means

$$\sum_{j=0}^{n-1} c_j \alpha^{ij} = 0 \qquad i = 1, \dots, n-k$$

But this means that $\alpha^1, \dots, \alpha^{n-k}$ are roots of $c(x)$, so

$$g(x) = \prod_{i=1}^{n-k} (x - \alpha^i) \Big| c(x)$$

for every code polynomial.

The main coefficient of $g(x)$ is 1, so it is the generator polynomial of the RS code.

Reed-Solomon codes

For a codeword $c = (c_0 c_1 \dots c_n)$, $cH^T = 0$ means

$$\sum_{j=0}^{n-1} c_j \alpha^{ij} = 0 \qquad i = 1, \dots, n-k$$

But this means that $\alpha^1, \dots, \alpha^{n-k}$ are roots of $c(x)$, so

$$g(x) = \prod_{i=1}^{n-k} (x - \alpha^i) \Big| c(x)$$

for every code polynomial.

The main coefficient of $g(x)$ is 1, so it is the generator polynomial of the RS code.

It also follows directly that RS codes with $n = q - 1$ are indeed cyclic.

Reed-Solomon codes

The fact that every codeword is a multiple of $g(x)$ means that it is possible to assign the codewords to the message vectors using

$$c(x) = u(x)g(x).$$

Reed-Solomon codes

The fact that every codeword is a multiple of $g(x)$ means that it is possible to assign the codewords to the message vectors using

$$c(x) = u(x)g(x).$$

Example. For the usual C(4,2) RS code, the codeword assigned to the message vector $u = (1\,2)$ is:

$$u = (1\,2) \;\rightarrow\; u(x) = 1 + 2x$$
$$c(x) = (1 + 2x)(3 + 4x + x^2) = 3 + 4x^2 + 2x^3 \;\rightarrow\; c = (3, 0, 4, 2)$$

Reed-Solomon codes

The fact that every codeword is a multiple of $g(x)$ means that it is possible to assign the codewords to the message vectors using

$$c(x) = u(x)g(x).$$

Example. For the usual C(4,2) RS code, the codeword assigned to the message vector $u = (1\,2)$ is:

$$u = (1\,2) \;\rightarrow\; u(x) = 1 + 2x$$
$$c(x) = (1 + 2x)(3 + 4x + x^2) = 3 + 4x^2 + 2x^3 \;\rightarrow\; c = (3, 0, 4, 2)$$

This assignment is different from $u \rightarrow c = uG$. It is not systematic either, but polynomial multiplication can be computed very efficiently (architecture coming soon).

Reed-Solomon codes – assignment I

Example. The C(4,2) RS code over GF(5) using the primitive element 2, codeword assignment based on $u \to c = uG$:

$(0\,0) \to (0\,0\,0\,0)$	$(2\,0) \to (2\,2\,2\,2)$	$(4\,0) \to (4\,4\,4\,4)$
$(0\,1) \to (1\,2\,4\,3)$	$(2\,1) \to (3\,4\,1\,0)$	$(4\,1) \to (0\,1\,3\,2)$
$(0\,2) \to (2\,4\,3\,1)$	$(2\,2) \to (4\,1\,0\,3)$	$(4\,2) \to (1\,3\,2\,0)$
$(0\,3) \to (3\,1\,2\,4)$	$(2\,3) \to (0\,3\,4\,1)$	$(4\,3) \to (2\,0\,1\,3)$
$(0\,4) \to (4\,3\,1\,2)$	$(2\,4) \to (1\,0\,3\,4)$	$(4\,4) \to (3\,2\,0\,1)$
$(1\,0) \to (1\,1\,1\,1)$	$(3\,0) \to (3\,3\,3\,3)$	
$(1\,1) \to (2\,3\,0\,4)$	$(3\,1) \to (4\,0\,2\,1)$	
$(1\,2) \to (3\,0\,4\,2)$	$(3\,2) \to (0\,2\,1\,4)$	
$(1\,3) \to (4\,2\,3\,0)$	$(3\,3) \to (1\,4\,0\,2)$	
$(1\,4) \to (0\,4\,2\,3)$	$(3\,4) \to (2\,1\,4\,0)$	

Reed-Solomon codes – assignment II

Example. The C(4,2) RS code over GF(5) using the primitive element 2, codeword assignment based on $c(x) = u(x)g(x)$:

$(0\,0) \to (0\,0\,0\,0)$	$(2\,0) \to (0\,1\,3\,2)$	$(4\,0) \to (0\,2\,1\,4)$
$(0\,1) \to (3\,4\,1\,0)$	$(2\,1) \to (3\,0\,4\,2)$	$(4\,1) \to (3\,1\,2\,4)$
$(0\,2) \to (1\,3\,2\,0)$	$(2\,2) \to (1\,4\,0\,2)$	$(4\,2) \to (1\,0\,3\,4)$
$(0\,3) \to (4\,2\,3\,0)$	$(2\,3) \to (4\,3\,1\,2)$	$(4\,3) \to (4\,4\,4\,4)$
$(0\,4) \to (2\,1\,4\,0)$	$(2\,4) \to (2\,2\,2\,2)$	$(4\,4) \to (2\,3\,0\,4)$
$(1\,0) \to (0\,3\,4\,1)$	$(3\,0) \to (0\,4\,2\,3)$	
$(1\,1) \to (3\,2\,0\,1)$	$(3\,1) \to (3\,3\,3\,3)$	
$(1\,2) \to (1\,1\,1\,1)$	$(3\,2) \to (1\,2\,4\,3)$	
$(1\,3) \to (4\,0\,2\,1)$	$(3\,3) \to (4\,1\,0\,3)$	
$(1\,4) \to (2\,4\,3\,1)$	$(3\,4) \to (2\,0\,1\,3)$	

Reed-Solomon codes – assignment III

Example. The C(4,2) RS code over GF(5) using the primitive element 2, systematic codeword assignment:

$(0\,0) \to (0\,0\,0\,0)$	$(2\,0) \to (2\,0\,1\,3)$	$(4\,0) \to (4\,0\,2\,1)$
$(0\,1) \to (0\,1\,3\,2)$	$(2\,1) \to (2\,1\,4\,0)$	$(4\,1) \to (4\,1\,0\,3)$
$(0\,2) \to (0\,2\,1\,4)$	$(2\,2) \to (2\,2\,2\,2)$	$(4\,2) \to (4\,2\,3\,0)$
$(0\,3) \to (0\,3\,4\,1)$	$(2\,3) \to (2\,3\,0\,4)$	$(4\,3) \to (4\,3\,1\,2)$
$(0\,4) \to (0\,4\,2\,3)$	$(2\,4) \to (2\,4\,3\,1)$	$(4\,4) \to (4\,4\,4\,4)$
$(1\,0) \to (1\,0\,3\,4)$	$(3\,0) \to (3\,0\,4\,2)$	
$(1\,1) \to (1\,1\,1\,1)$	$(3\,1) \to (3\,1\,2\,4)$	
$(1\,2) \to (1\,2\,4\,3)$	$(3\,2) \to (3\,2\,0\,1)$	
$(1\,3) \to (1\,3\,2\,0)$	$(3\,3) \to (3\,3\,3\,3)$	
$(1\,4) \to (1\,4\,0\,2)$	$(3\,4) \to (3\,4\,1\,0)$	

Systematic generation

In general, a cyclic linear code with generator polynomial $g(x) = g_0 + g_1 x + \cdots + g_{n-k} x^{n-k}$ can be generated systematically by the generator matrix

$$G = \begin{bmatrix} 1 & g_1' & g_2' & \dots & g_{n-k-1}' & g_{n-k}' & 0 & \dots & 0 & 0 \\ 0 & 1 & g_1' & \dots & g_{n-k-2} & g_{n-k-1}' & g_{n-k}' & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & & \vdots \\ 0 & 0 & 0 & \dots & 1 & g_1' & g_2' & \dots & g_{n-k}' & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 & g_1' & \dots & g_{n-k-1}' & g_{n-k}' \end{bmatrix}$$

where $g_i' = g_i / g_0$. (g_0 cannot be 0 since $g(x) | x^n - 1$.)

Systematic generation

In general, a cyclic linear code with generator polynomial $g(x) = g_0 + g_1 x + \cdots + g_{n-k} x^{n-k}$ can be generated systematically by the generator matrix

$$G = \begin{bmatrix} 1 & g'_1 & g'_2 & \ldots & g'_{n-k-1} & g'_{n-k} & 0 & \ldots & 0 & 0 \\ 0 & 1 & g'_1 & \ldots & g_{n-k-2} & g'_{n-k-1} & g'_{n-k} & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & & \vdots \\ 0 & 0 & 0 & \ldots & 1 & g'_1 & g'_2 & \ldots & g'_{n-k} & 0 \\ 0 & 0 & 0 & \ldots & 0 & 1 & g'_1 & \ldots & g'_{n-k-1} & g'_{n-k} \end{bmatrix}$$

where $g'_i = g_i / g_0$. (g_0 cannot be 0 since $g(x) | x^n - 1$.) For code polynomials, the formula for systematic assignment is

$$c(x) = u(x)x^{n-k} - (u(x)x^{n-k} \mod g(x)).$$

Binary linear cyclic codes

The result that linear cyclic codes can be generated using a generator polynomial is valid for binary codes as well.

Binary linear cyclic codes

The result that linear cyclic codes can be generated using a generator polynomial is valid for binary codes as well.

So are any of the binary codes we have seen previously cyclic?

Binary linear cyclic codes

The result that linear cyclic codes can be generated using a generator polynomial is valid for binary codes as well.

So are any of the binary codes we have seen previously cyclic?

- repeater codes are clearly cyclic; the $\mathrm{C}(n, 1)$ repeater code has generator polynomial $g(x) = 1 + x + \cdots + x^{n-1}$.

Binary linear cyclic codes

The result that linear cyclic codes can be generated using a generator polynomial is valid for binary codes as well.

So are any of the binary codes we have seen previously cyclic?

- repeater codes are clearly cyclic; the C$(n, 1)$ repeater code has generator polynomial $g(x) = 1 + x + \cdots + x^{n-1}$.
- Hamming codes are also cyclic (at least for some orderings of the columns of the parity check matrix H). Generator polynomials:
  - C(7,4): $g(x) = 1 + x + x^3$;
  - C(15,11): $g(x) = 1 + x + x^4$ or $g(x) = 1 + x^3 + x^4$;
  - C(31,26): $g(x) = 1 + x^2 + x^5$.
  - etc.

Binary linear cyclic codes

The result that linear cyclic codes can be generated using a generator polynomial is valid for binary codes as well.

So are any of the binary codes we have seen previously cyclic?

- repeater codes are clearly cyclic; the C$(n, 1)$ repeater code has generator polynomial $g(x) = 1 + x + \cdots + x^{n-1}$.
- Hamming codes are also cyclic (at least for some orderings of the columns of the parity check matrix H). Generator polynomials:
  - C(7,4): $g(x) = 1 + x + x^3$;
  - C(15,11): $g(x) = 1 + x + x^4$ or $g(x) = 1 + x^3 + x^4$;
  - C(31,26): $g(x) = 1 + x^2 + x^5$.
  - etc.
- the C(23,12) Golay code is also cyclic with either $g(x) = 1 + x^2 + x^4 + x^5 + x^6 + x^{10} + x^{11}$ or $g(x) = 1 + x^1 + x^5 + x^6 + x^7 + x^9 + x^{11}$.

Binary linear cyclic codes

The result that linear cyclic codes can be generated using a generator polynomial is valid for binary codes as well.

So are any of the binary codes we have seen previously cyclic?

- repeater codes are clearly cyclic; the $C(n, 1)$ repeater code has generator polynomial $g(x) = 1 + x + \cdots + x^{n-1}$.
- Hamming codes are also cyclic (at least for some orderings of the columns of the parity check matrix H). Generator polynomials:
  - C(7,4): $g(x) = 1 + x + x^3$;
  - C(15,11): $g(x) = 1 + x + x^4$ or $g(x) = 1 + x^3 + x^4$;
  - C(31,26): $g(x) = 1 + x^2 + x^5$.
  - etc.
- the C(23,12) Golay code is also cyclic with either $g(x) = 1 + x^2 + x^4 + x^5 + x^6 + x^{10} + x^{11}$ or $g(x) = 1 + x^1 + x^5 + x^6 + x^7 + x^9 + x^{11}$.

Not cyclic: Hadamard.

Systematic code generation

Code generation with generator polynomials

What do we gain by using polynomials to describe codes?

Code generation with generator polynomials

What do we gain by using polynomials to describe codes?

- Polynomials are smaller than matrices. E.g. the C(6,4) RS code over GF(7) generated by the primitive element 5 has

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 5 & 4 & 6 & 2 & 3 \\ 1 & 4 & 2 & 1 & 4 & 2 \\ 1 & 6 & 1 & 6 & 1 & 6 \end{bmatrix}$$

  and

$$g(x) = 4 + 2x + 3x^2 + 6x^3 + x^4.$$

Code generation with generator polynomials

What do we gain by using polynomials to describe codes?

- Polynomials are smaller than matrices. E.g. the C(6,4) RS code over GF(7) generated by the primitive element 5 has

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 5 & 4 & 6 & 2 & 3 \\ 1 & 4 & 2 & 1 & 4 & 2 \\ 1 & 6 & 1 & 6 & 1 & 6 \end{bmatrix}$$

  and

$$g(x) = 4 + 2x + 3x^2 + 6x^3 + x^4.$$

- The computational cost of polynomial multiplication is comparable to matrix-vector multiplication.

Code generation with generator polynomials

What do we gain by using polynomials to describe codes?

- Polynomials are smaller than matrices. E.g. the C(6,4) RS code over GF(7) generated by the primitive element 5 has

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 5 & 4 & 6 & 2 & 3 \\ 1 & 4 & 2 & 1 & 4 & 2 \\ 1 & 6 & 1 & 6 & 1 & 6 \end{bmatrix}$$

  and

$$g(x) = 4 + 2x + 3x^2 + 6x^3 + x^4.$$

- The computational cost of polynomial multiplication is comparable to matrix-vector multiplication.
- Efficient decoding methods for polynomials (coming soon).

Brief summary of code polynomials (so far)

Every cyclic linear $C(n,k)$ code over $GF(q)$ has a generator polynomial $g(x)$ over $GF(q)$ such that

- $c(x)$ is a code polynomial $\iff g(x)|c(x)$;
- $g(x)|x^n - 1$;
- $\deg(g(x)) = n - k$, and
- the main coefficient of $g(x)$ is $g_{n-k} = 1$.

Brief summary of code polynomials (so far)

Every cyclic linear $C(n, k)$ code over $GF(q)$ has a generator polynomial $g(x)$ over $GF(q)$ such that

- $c(x)$ is a code polynomial $\iff g(x)|c(x)$;
- $g(x)|x^n - 1$;
- $\deg(g(x)) = n - k$, and
- the main coefficient of $g(x)$ is $g_{n-k} = 1$.

We can generate codewords from message vectors with

$$c(x) = u(x)g(x).$$

This gives a different $u \to c$ assignment than matrix-vector multiplication.

Decoding with generator polynomials

For encoding in polynomial form, we will stick to using

$$c(x) = u(x)g(x).$$

Next we prepare for decoding using code polynomials.

Decoding with generator polynomials

For encoding in polynomial form, we will stick to using

$$c(x) = u(x)g(x).$$

Next we prepare for decoding using code polynomials.

The syndrome polynomial assigned to a received code polynomial $v(x)$ is

$$s(x) = v(x) \mod g(x)$$

A received polynomial $v(x)$ is a codeword $\iff s(x) = 0$.

Error trapping algorithm

Next we present the Error trapping algorithm for decoding RS codes. It has more limited error correction capability than the Error locator algorithm, but it is much faster at any parameter setup.

Error trapping algorithm

Next we present the Error trapping algorithm for decoding RS codes. It has more limited error correction capability than the Error locator algorithm, but it is much faster at any parameter setup.

Notably the Error trapping algorithm can correct $\left\lfloor \frac{n-k}{2} \right\rfloor$ errors for a $C(n, k)$ RS code as long as all of the errors fall close to each other.

Error trapping algorithm

Next we present the Error trapping algorithm for decoding RS codes. It has more limited error correction capability than the Error locator algorithm, but it is much faster at any parameter setup.

Notably the Error trapping algorithm can correct $\left\lfloor \frac{n-k}{2} \right\rfloor$ errors for a $C(n, k)$ RS code as long as all of the errors fall close to each other.

This restriction is outside the usual definition of error correction capabilities, but for some physical channels (not the q-ary symmetric channel), errors typically occur in bursts, and the Error trapping algorithm is very relevant in such situations.

Error trapping algorithm

Assume we have a C(n, k) RS code over GF(q), and the error vector e is such that

- $w(e) \leq \frac{n-k}{2}$;
- all errors in e occur within an interval of $(n-k)$ consecutive digits.

Error trapping algorithm

Assume we have a $C(n, k)$ RS code over $GF(q)$, and the error vector e is such that

- $w(e) \leq \frac{n-k}{2}$;
- all errors in e occur within an interval of $(n-k)$ consecutive digits.

The received polynomial is

$$v(x) = u(x)g(x) + e(x)$$

due to the channel model.

Error trapping algorithm

Assume we have a $C(n, k)$ RS code over $GF(q)$, and the error vector e is such that

- $w(e) \leq \frac{n-k}{2}$;
- all errors in e occur within an interval of $(n-k)$ consecutive digits.

The received polynomial is

$$v(x) = u(x)g(x) + e(x)$$

due to the channel model.

Dividing the polynomial $v(x)$ by $g(x)$ yields

$$v(x) = a(x)g(x) + r(x)$$

where $\deg(r(x)) < \deg(g(x)) = n - k$.

Error trapping algorithm

Assume we have a $C(n, k)$ RS code over $GF(q)$, and the error vector e is such that

- $w(e) \leq \frac{n-k}{2}$;
- all errors in e occur within an interval of $(n-k)$ consecutive digits.

The received polynomial is

$$v(x) = u(x)g(x) + e(x)$$

due to the channel model.

Dividing the polynomial $v(x)$ by $g(x)$ yields

$$v(x) = a(x)g(x) + r(x)$$

where $\deg(r(x)) < \deg(g(x)) = n - k$.

Do the two equations imply $r(x) = e(x)$?

Error trapping algorithm

No, $r(x) = e(x)$ is not implied. We know that $\deg(r(x)) < n - k$, but $\deg(e(x))$ can be higher, up to $n - 1$.

Error trapping algorithm

No, $r(x) = e(x)$ is not implied. We know that $\deg(r(x)) < n - k$, but $\deg(e(x))$ can be higher, up to $n - 1$.

However, we assumed that e has all errors within coordinates $[i, i + n - k - 1]$ for some i. Assume for a moment that the value of i is available. Then applying S^{-i} in advance for the previous calculations shifts the nonzero terms of $e(x)$ to the front; in other words, $\deg(S^{-i}e(x)) < n - k$.

Error trapping algorithm

No, $r(x) = e(x)$ is not implied. We know that $\deg(r(x)) < n - k$, but $\deg(e(x))$ can be higher, up to $n - 1$.

However, we assumed that e has all errors within coordinates $[i, i + n - k - 1]$ for some i. Assume for a moment that the value of i is available. Then applying S^{-i} in advance for the previous calculations shifts the nonzero terms of $e(x)$ to the front; in other words, $\deg(S^{-i}e(x)) < n - k$.

Then

$$S^{-i}v(x) = S^{-i}u(x)g(x) + S^{-i}e(x)$$
$$S^{-i}v(x) = S^{-i}a(x)g(x) + r(x),$$

and now $S^{-i}e(x) = r(x)$ is implied since $S^{-i}e(x) - r(x)$ is a codeword, so it is a multiple of $g(x)$, but $\deg(S^{-i}e(x) - r(x)) < \deg(g(x))$, so $S^{-i}e(x) - r(x)$ must be 0.

Error trapping algorithm

Based on the previous calculation, if we knew the position i where the errors start in $e(x)$, we could do the following:

- compute the polynomial division $S^{-i}v(x) = S^{-i}a(x)g(x) + r(x)$, and
- the detected error polynomial is $e'(x) = S^{i}r(x)$.

Error trapping algorithm

Based on the previous calculation, if we knew the position i where the errors start in $e(x)$, we could do the following:

- compute the polynomial division $S^{-i}v(x) = S^{-i}a(x)g(x) + r(x)$, and
- the detected error polynomial is $e'(x) = S^{i}r(x)$.

The issue is that the position i is not known in advance.

Error trapping algorithm

Based on the previous calculation, if we knew the position i where the errors start in $e(x)$, we could do the following:

- compute the polynomial division $S^{-i}v(x) = S^{-i}a(x)g(x) + r(x)$, and
- the detected error polynomial is $e'(x) = S^{i}r(x)$.

The issue is that the position i is not known in advance.

So how can we obtain i?

Error trapping algorithm

Based on the previous calculation, if we knew the position i where the errors start in $e(x)$, we could do the following:

- compute the polynomial division $S^{-i}v(x) = S^{-i}a(x)g(x) + r(x)$, and
- the detected error polynomial is $e'(x) = S^{i}r(x)$.

The issue is that the position i is not known in advance.

So how can we obtain i?

Let i be any position (not necessarily where the errors in $e(x)$ start), and compute the polynomial division

$$S^{-i}v(x) = S^{-i}a(x)g(x) + r^{(i)}(x)$$

anyway.

Error trapping algorithm

Based on the previous calculation, if we knew the position i where the errors start in $e(x)$, we could do the following:

- compute the polynomial division $S^{-i}v(x) = S^{-i}a(x)g(x) + r(x)$, and
- the detected error polynomial is $e'(x) = S^{i}r(x)$.

The issue is that the position i is not known in advance.

So how can we obtain i?

Let i be any position (not necessarily where the errors in $e(x)$ start), and compute the polynomial division

$$S^{-i}v(x) = S^{-i}a(x)g(x) + r^{(i)}(x)$$

anyway.

The polynomial $S^{-i}e(x) - r^{(i)}(x) = S^{-i}v(x) - r^{(i)}(x)$ is a code polynomial, which means that either

$$S^{-i}v(x) - r^{(i)}(x) = 0 \quad \text{or} \quad w(S^{-i}v(x) - r^{(i)}(x)) \geq n - k + 1$$

because the RS code has minimal codeword distance $n - k + 1$.

Error trapping algorithm

Based on this, the Error trapping algorithm is the following:

- For each value of i from 0 to $n-1$, compute the polynomial division $S^{-i}v(x) = S^{-i}a(x)g(x) + r^{(i)}(x)$.
- Compute $w(S^{-i}v(x) - r^{(i)}(x))$.
- If $w(S^{-i}v(x) - r^{(i)}(x)) \geq n-k+1$, move on to the next value of i.
- If $w(S^{-i}v(x) - r^{(i)}(x)) < n-k+1$, then we stop; $e'(x) = S^{i}r^{(i)}(x)$ must hold, and so the detected error polynomial is
$$e'(x) = S^{i}r^{(i)}(x).$$

Error trapping algorithm

Based on this, the Error trapping algorithm is the following:

- For each value of i from 0 to $n-1$, compute the polynomial division $S^{-i}v(x) = S^{-i}a(x)g(x) + r^{(i)}(x)$.
- Compute $w(S^{-i}v(x) - r^{(i)}(x))$.
- If $w(S^{-i}v(x) - r^{(i)}(x)) \geq n-k+1$, move on to the next value of i.
- If $w(S^{-i}v(x) - r^{(i)}(x)) < n-k+1$, then we stop; $e'(x) = S^i r^{(i)}(x)$ must hold, and so the detected error polynomial is

$$e'(x) = S^i r^{(i)}(x).$$

If $e(x)$ indeed contains all $\leq \left\lfloor \frac{n-k}{2} \right\rfloor$ errors within an interval of length $\leq n-k$, then the algorithm is guaranteed to stop for at least one choice of i. (It is possible that several i positions are good, then we can use either of them.)