8.1 Error-Decoding RS Codes

8.1.1 Welch-Berlekamp Algorithm

Recall from week 7 the received vector $\vec{y} = \vec{c} + \vec{e}$ where $Hw(\vec{e}) \leq \lfloor \frac{n-k}{2} \rfloor = \tau$. Also recall that $\alpha_1, \alpha_2, \ldots, \alpha_n$ are the code locators. We produce the following polynomial:

$$P(x, y) = yE(x) - M(x)E(x)$$

Where:

$$N(x) = M(x)E(x)$$
$$E(x) = \prod_{\alpha_i \in S} (x - \alpha_i)$$

Recall that $S$ is the set of locations of errors and $|S| \leq \lfloor \frac{n-k}{2} \rfloor$. Thus, $P(\alpha_i, y_i) = 0$ for $i = 1, 2, \ldots, n$. The goal of the Welch-Berlekamp method is to find polynomials $E_1(x)$ and $N_1(x)$ that satisfy the following:

1. $y_iE(\alpha_i) = N(\alpha_i), i = 1, 2, \ldots, n;$
2. $\deg(E(x)) \leq \tau$;
3. $\deg(N(x)) \leq \tau + k - 1$.

Note that $N(X)$ has $\tau + k - 1$ variables and has the following form: $N(x) = N_0 + N_1 x + \cdots + N_{\tau+k-1} x^{\tau+k-1}$. Similarly, $E(X)$ has $\tau$ variables and has the following form: $E(x) = E_0 + E_1 x + \cdots + E_{\tau-1} x^{\tau-1} + E_{\tau} x^\tau$.

Statement 1 forms $n$ different equations where there are $2\tau + k + 1$ unknowns. Since $n < 2\tau + k + 1$, these equations are solvable. There exists at least one solution since $E(x)$ and $N(x) = M(x)E(x)$ is a solution to all three statements.

In fact, there might exist multiple solutions. However, let $N_1(x), E_1(x)$ be a solution to statements 1, 2 and 3. Additionally, let $N_2(x), E_2(x)$ be another solution to these statements. We show that

$$\frac{N_1(x)}{E_1(x)} = \frac{N_2(x)}{E_2(x)},$$

which implies that the message polynomial can be uniquely recovered as $M(x) = \frac{N_1(x)}{E_1(x)}$.

We prove this by creating the polynomial $R_i(x) = N_i(x) - M(x)E(x)$ where $N_i(x)$ and $E_i(x)$ are the outputs of the algorithm and $M(x)$ is the actual message. We know that $\deg(R_i(x)) \leq \tau + k - 1$. Furthermore, we
know that $M(\alpha_i) = y_i$ at $n - \tau$ points, so then $R_i(\alpha_j) = 0$ for at least $n - \tau$ evaluation points. Note that $n - \tau > \tau + k - 1$, thus we have $R_i(x) = 0$ for all $x$. That is $R_i(x)$ is the zero polynomial. This proves that $M(x) = \frac{N(x)}{E_i(x)}$.

**8.1.2 PGZ Decoder**

The PGZ decoder differs from that of Welch-Berlekamp in that it uses the syndrome of the received code in order to decode it. The main idea of the PGZ decoder was discovered independently by Prony in 1795. Note that the PGZ decoder method only works for conventional Narrow-sense RS codes. Recall that the generator matrix and parity check matrix for such a code are as follows:

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

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

Again, our received vector is $\tilde{y} = \tilde{c} + \tilde{e}$, or $y(x) = c(x) + e(x)$ in a polynomial viewpoint.

Recall that the syndrome of a code is as follows:

$$\tilde{S} = \begin{bmatrix} S_0 \\ S_1 \\ \vdots \\ S_{n-k-1} \end{bmatrix} = He^T + He^T = He^T$$

Thus, from the polynomial viewpoint, $S_i = e(\alpha^{i+1})$, $i = 0, 1, \ldots, n - k - 1$. We know that $Hw(\tilde{e}) \leq \tau$, so then $e(x)$ is a $\tau$-sparse polynomial of degree $n - 1$. We know that the syndrome is:

$$\tilde{S} = \begin{bmatrix} e(\alpha) \\ e(\alpha^2) \\ \vdots \\ e(\alpha^{2\tau}) \end{bmatrix}$$

Our goal then is to find a $\tau$-sparse polynomial $e(x)$ from $2\tau$ evaluations at $\alpha, \alpha^2, \ldots, \alpha^{2\tau}$.

**8.1.2.1 Sparse polynomial interpretation**

Example: For $\tau = 2$, $e(x) = d_1x^{i_1} + d_2x^{i_2}$ Thus, the syndromes are as follows:

$s_0 = e_1\alpha^{i_1} + e_2\alpha^{i_2}$
$s_1 = e_1\alpha^{2i_1} + e_2\alpha^{2i_2}$
$s_2 = e_1\alpha^{3i_1} + e_2\alpha^{3i_2}$
$s_3 = e_1\alpha^{4i_1} + e_2\alpha^{4i_2}$
We define $\alpha^i_1 := b_1$, $\alpha^i_2 := b_2$, $e_i := d_1$ and $e_i := d_2$. This gives us:

$$
\begin{align*}
    s_0 &= d_1 b_1 + d_2 b_2 \\
    s_1 &= d_1 b_1^2 + d_2 b_2^2 \\
    s_2 &= d_1 b_1^3 + d_2 b_2^3 \\
    s_3 &= d_1 b_1^4 + d_2 b_2^4
\end{align*}
$$

Define error locator polynomial $\Lambda(x)$:

$$
\Lambda(x) = (x - b_1)(x - b_2) = x + \Lambda_1 x + \Lambda_2
$$

Clearly,

$$
\begin{align*}
    \Lambda(b_1) &= 0 \\
    \Lambda(b_2) &= 0
\end{align*}
$$

Thus we can set up the following equation:

$$
\begin{align*}
    d_1 b_1 \Lambda(b_1) + d_2 b_2 \Lambda(b_2) &= 0 \\
    d_1 b_1 [b_1 + \Lambda_1 b_1 + \Lambda_2] + d_2 b_2 [b_2 + \Lambda_1 b_2 + \Lambda_2] &= 0 \\
    \Lambda_0 [d_1 b_1 + d_2 b_2] + \Lambda_1 [d_1 b_1^2 + d_2 b_2^2] + [d_1 b_1^3 + d_2 b_2^3] &= 0 \\
    \Lambda_0 s_0 + \Lambda_1 s_1 + s_2 &= 0
\end{align*}
$$

Similarly we can write:

$$
\begin{align*}
    b_1^2 d_1 \Lambda(b_1) + b_2^2 d_2 \Lambda(b_2) &= 0 \\
    \Lambda_0 s_1 + \Lambda_1 s_2 + s_3 &= 0
\end{align*}
$$

This can be written in matrix form as follows:

$$
\begin{bmatrix}
    s_0 & s_1 \\
    s_1 & s_2
\end{bmatrix}
\begin{bmatrix}
    \Lambda_0 \\
    \Lambda_1
\end{bmatrix}
= -
\begin{bmatrix}
    s_2 \\
    s_3
\end{bmatrix}
$$

In general, we have:

$$
\begin{bmatrix}
    s_0 & s_1 & \ldots & s_{t-1} \\
    s_1 & s_2 & \ldots & s_t \\
    \vdots & \vdots & \ddots & \vdots \\
    s_{t-1} & s_t & \ldots & s_{2t-2}
\end{bmatrix}
\begin{bmatrix}
    \Lambda_0 \\
    \Lambda_1 \\
    \vdots \\
    \Lambda_{t-1}
\end{bmatrix}
= -
\begin{bmatrix}
    s_t \\
    s_{t+1} \\
    \vdots \\
    s_{2t-1}
\end{bmatrix}
$$

This is called the **Key equation**.

### 8.1.2.2 PGZ Algorithm Summary

1. Solve Key equation, find $\Lambda_0, \Lambda_1, \ldots, \Lambda_{t-1}$.
2. Find roots of $\Lambda(x) = \Lambda_0 + \Lambda_1 x + \cdots + \Lambda_{k-1} x^{k-1} + x^t$: $\alpha^{i_1}, \alpha^{i_2}, \ldots, \alpha^{i_t}$
   (Further reading: “Chien search”)
3. Note $c(x) = \sum_{j=1}^t c_{ij} x^{i_j}$.
   Replace erroneous locations $i_1, i_2, \ldots, i_t$ with erasures and then use Langrange interpolation.
   (Further reading: Forney’s algorithm)
8.1.3 Fourier Transform Interpretation of PGZ

Recall the following forms for received vector polynomial $y(x)$, syndromes $s_i$, and error locator polynomial $\Lambda(x)$:

$$y(x) = c(x) + e(x)$$

Assume that the erroneous co-ordinates are $S = \{s_1, s_2, \ldots, s_t\} \subset \{0, 1, 2, \ldots, n - 1\}$. Note that $e(x) = e_0 + e_1x + e_2x^2 + \ldots + e_{n-1}x^{n-1} = \sum_{i \in S} e_i x^i$.

$$s_i = e(\alpha^i) \quad \text{for} \quad i = 1, 2, \ldots, n-k$$

$$\Lambda(x) = \prod_{i=1}^{t}(x - \alpha^{t_i}) = \Lambda_0 + \Lambda_1 x + \cdots + \Lambda_{t-1} x^{t-1} + x^t$$

Let $\vec{E} = (E_0, E_1, \ldots, E_{n-1})$ be the inverse Fourier transform of $\vec{e} = (e_0, e_1, \ldots, e_{n-1})$. Note that $s_i = E_{n-i}$, $i = 1, 2, \ldots, n-k$.

Let

$$\vec{\Lambda} = \begin{bmatrix} \Lambda_0 \\ 0 \\ \vdots \\ \Lambda_t \\ \Lambda_{t-1} \\ \vdots \\ \Lambda_1 \end{bmatrix}.$$ 

Let $\vec{\lambda} = (\lambda_0, \lambda_1, \ldots, \lambda_{n-1})$ be the Inverse Fourier Transform of $\vec{\Lambda}$. That is

$$\lambda_i = \sum_{j=n-1}^{n-1-1} \Lambda_{n-1-j} \alpha^{i(n-j)} = \Lambda(\alpha^i)$$

The above equation implies that $\beta = \alpha^i$ is a root of $\Lambda(x)$ if and only if $\lambda_i = 0$. In other words, $i \in S \Rightarrow \lambda_i = 0$. Therefore $\lambda_i e_i = 0$ for all $i = 0, 1, 2, \ldots, n-1$. That is $\vec{\lambda} \cdot \vec{e} = 0$. Multiplication in time domain corresponds to cyclic convolution in frequency domain. Thus we have $\vec{\Lambda} \ast \vec{E} = \vec{0}$. Note that this is a set of $n$ equations in $t$ unknowns of $\vec{\Lambda}$, the $n-k$ known co-ordinates of $\vec{E}$ and the remaining $k$ unknown co-ordinates of $\vec{E}$. The Key equation described previously simply expresses the $(n-k)/2$ equations that are exclusively $n-k$ known co-ordinates of $\vec{E}$. It is instructive to observe that the toeplitz structure of the convolution matrix appears in the Key equation.

8.2 Cyclic Codes

Definition 8.1 (Cyclic Code)

$C$ is a cyclic code if $\vec{c} = (c_0, c_1, c_2, \ldots, c_{n-1}) \in C \iff (c_{n-1}, c_0, c_1, \ldots, c_{n-1}) \in C$. 
8.2.1 Polynomial Interpretation of Cyclic codes

$C$ is a collection of polynomials that is a subset of $\mathbb{F}[x]/(x^n - 1)$. Let $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$ and $c(x) = c_0 + c_1 x + \cdots + c_{n-1} x^{n-1}$. Note that $x \cdot c(x) = c_{n-1} + c_0 x + \cdots + c_{n-2} x^{n-1}$ when using mod($x^n - 1$). Thus, another definition of cyclic codes is $c(x) \in C$ such that $x \cdot c(x) \in C$.

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

$C$ is cyclic implies: $c(x) \in C \iff x \cdot c(x) \in C$.

Lemma 8.2 Let $C \in \mathbb{F}[x]/(x^n-1)$ be a linear cyclic code. If $c(x) \in C$, then $u(x)c(x) \in C$ for all $u(x) \in \mathbb{F}[x]$.

Proof: Suppose $u(x) = u_0 + u_1 x + \cdots + u_d x^d$.

From $c(x) \in C$ and $x \cdot c(x) \in C$, we know:

\[ u_0 c(x) \in C \quad \text{(linear)} \]
\[ u_1 x c(x) \in C \quad \text{(linear & cyclic)} \]
\[ \vdots \]
\[ d_d x^d c(x) \in C \quad \text{(linear & cyclic)} \]

Thus $u(x)c(x) = (u_0 + u_1 x + \cdots + u_d x^d)c(x) \in C$. \hfill \qed

Lemma 8.3 Let $C$ be a linear cyclic code. Then there exists unique monic polynomial $g(x) \in C$ such that 1) $u(x)g(x) \in C$ for all $u(x) \in \mathbb{F}[x]$, where multiplication is performed modulo $x^n - 1$; and 2) $c(x) \in C \Rightarrow c(x) = u(x)g(x)$ for some $u(x)$.

Proof: Let $g(x)$ be a non-zero monic polynomial with minimal degree in $c(x)$. We claim that $g(x)$ satisfies 1) and 2).

$g(x)$ satisfies 1) because of Lemma 8.2.

To show 2), suppose $c(x) = g(x)u(x) + r(x)$ where $\deg(r(x)) < \deg(g(x))$.

\[ r(x) = c(x) - g(x)u(x) \in C \text{ since } C \text{ is linear code.} \]

$\deg(r(x)) < \deg(g(x)) \Rightarrow r(x) = 0$ since $g(x)$ is the minimum degree non-zero polynomial. The above equation also implies that $g(x)$ is unique. Specifically, if $g_1(x)$ is another monic polynomial with minimal degree, the above equation implies that $g_1(x) = u(x)g(x)$ where $u$ is a constant. However, both $g(x)$ and $g_1(x)$ are monic, implying that $u = 1$, hence $g(x) = g_1(x)$. \hfill \qed

Definition 8.4 (Generator Polynomial)

$g(x)$ defined in Lemma 8.3 is the generator polynomial of code $C$.

Lemma 8.5 $C$ is an $(n, k, d)$ linear cyclic code. Then its generator polynomial $g(x)$ has degree $n - k$.

Proof: We show that $C$ has dimension $n - \deg(g)$. Every unique polynomial $u(x)$ of degree less than $k$ leads to a unique codeword $c(x) = u(x)g(x)$. This implies that the dimension of $C$ is at least $n - \deg(g)$. Furthermore, since $g(x)$ is the unique monic polynomial of minimum degree in $C$, every codeword $c(x)$ has
Lemma 8.6

Let \( g \) be the generator polynomial.

Proof: Given the generator polynomial, \( g(x) = g_0 + g_1 x + \cdots + g_{n-k} x^{n-k} \) and the generator matrix is

\[
G = \begin{bmatrix}
g_0 & g_1 & g_2 & \cdots & g_{n-k} & 0 & 0 & \cdots & 0 & 0 \\
g_0 & g_1 & g_2 & \cdots & g_{n-k} & 0 & 0 & \cdots & 0 & 0 \\
\vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\
0 & 0 & 0 & \cdots & 0 & g_{n-k-1} & g_{n-k} \\
\end{bmatrix}
\]

Lemma 8.6 C is a cyclic code and \( g(x) \) is the generator polynomial of \( C \). Then \( g(x)(x^n - 1) \) in \( \mathbb{F}[x] \).

Proof: Suppose \( x^n - 1 = g(x)u(x) + r(x) \), we have

\[
g(x)u(x) = -r(x) \pmod{(x^n - 1)}
\]

Note that \( g(x) \) is the smallest degree non-zero polynomial of \( C \). We must have \( r(x) = 0 \).

Let \( h(x) = \frac{x^{n-1}}{g(x)} \), then \( h(x) \) is the parity check polynomial.

\[
h(x) = h_0 + h_1 x + \cdots + h_k x^k,
\]

and the parity check matrix is

\[
H = \begin{bmatrix}
h_k & h_{k-1} & h_{k-2} & \cdots & h_0 & 0 & \cdots & 0 & 0 \\
0 & h_k & h_{k-1} & \cdots & h_1 & h_0 & \cdots & 0 & 0 \\
\vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots & \vdots \\
0 & 0 & 0 & \cdots & 0 & h_2 & h_1 \\
\end{bmatrix}
\]

8.2.1.1 Examples of Cyclic Codes

1. Conventional Narrow-sense Reed-Solomon Code

\[ C \rightarrow (c_0, c_1, \ldots, c_{n-1}) \], or \( c(x) = c_0 + c_1 x + \cdots + c_{n-1} x^{n-1} \).

One can verify \( g(x) = (x - \alpha)(x - \alpha^2) \cdots (x - \alpha^{n-k}) \) is the generator polynomial.

2. Binary BCH Code

Let \( C \) be an \((n, k)\) primitive RS codes over \( \mathbb{F}_{2^m}^n \), where \( n = 2^m - 1 \).

\( C_{BCH} = (C \cap F_2^n) \subset F_2^n \), it is linear over \( F_2 \).

3. \((7, 4)\) Hamming code

\[
g(x) = x^3 + x + 1.
\]

\[
h(x) = x^4 + x^2 + x + 1.
\]

One can verify that \( g(x)h(x) = x^7 + 1 \). Its parity check matrix is

\[
H(x) = \begin{bmatrix}
1 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 1
\end{bmatrix}
\]

Recall that this is a \((7, 4)\) Hamming code.