In today’s lecture, we continue our study of error-correcting codes.

1 The Singleton Bound

We establish a simple trade-off between dimension and distance that any linear code satisfies.

**Theorem 1.1.** Any \([n, k, d]_q\) code must satisfy

\[ k + d \leq n + 1 \]  

**Proof:** Let \( C \) be an \([n, k, d]_q\) code. Let code \( C_1 \) be almost the same as \( C \) except with the first symbol removed from each codeword. The block size will now be of size \( n - 1 \). The Hamming distance will now be at least \( d - 1 \). This is because we are only removing one bit (one less possible differing bit) from the codewords produced by \( C \).

\[ C^1 \rightarrow [n - 1, k, \geq d - 1]_q \]  

We can now remove the first \( i \) symbols to produce code \( C^i \)

\[ C^i \rightarrow [n - i, k, \leq d - i] \]  

Choose \( i = d - 1 \)

\[ C^{d-1} \rightarrow [n - (d - 1), k, 1] \]

The block size must be greater than or equal to the dimension of the code which gives us the singleton bound.

\[ n - (d - 1) \geq k \]

\[ k + d \leq n + 1 \]

This makes sense conceptually because as the number of messages increases, the more codewords you need to pack into \( \mathbb{F}^n \), and thus the distance must decrease.

2 Welch-Berlekamp Algorithm

Recall the Reed Solomon code: given a message \( \vec{a} \in \mathbb{F}^k \), we view it as the polynomial \( P_\vec{a}(x) = \sum_{i=0}^{k-1} a_i x^i \). Fix \( S = \{ \varphi_1, \varphi_2, ... \varphi_n \} \), and the codeword is produced as follows

\[ a = (a_1, a_2, ..., a_{k-1}) \rightarrow (P_\vec{a}(\varphi_1), P_\vec{a}(\varphi_2), ..., P_\vec{a}(\varphi_n)) = c \in \mathbb{F}^n \]

The codeword is now sent through a noisy channel that makes \( e \) errors (i.e., changes at most \( e \) symbols).

\[ c \xrightarrow{\text{noisy channel}} c' \in \mathbb{F}^n \]
We can represent \( c' \) as a function \( f \) that evaluates to the corresponding symbols in \( c' \) on all the points in \( S \)

\[
f : \mathbb{F} \rightarrow \mathbb{F} \quad \forall i \in [n], \ f(\varphi_i) = c'_i
\]

A key definition towards developing the algorithm for error-correcting RS codes is the following:

**Definition 2.1.** An Error-Locator Polynomial is a polynomial \( E \) is such that

\[
E(x) = 0 \iff f(x) \neq P_{\vec{a}}(x)
\]

In other words, \( E \) has a root whenever \( f \) and \( P_{\vec{a}} \) differ.

**Observation 2.2.** There exists a polynomial \( E \) of degree equal to \( e \) (the number of errors).

**Proof:** Let \( \{\beta_1, \beta_2, ..., \beta_e\} \) be the locations of the errors (all \( x \) where \( f(x) \neq P_{\vec{a}}(x) \))

\[
E(x) = \prod_{i=1}^{e} (x - \beta_i)
\]

The following is a simple but key identity.

**Observation 2.3.** \( \forall x \in S, \ f(x)E(x) = P_{\vec{a}}(x)E(x) \)

When \( f(x) \) and \( P_{\vec{a}}(x) \) differ, \( E(x) = 0 \) so the equality holds. When \( f(x) \) and \( P_{\vec{a}}(x) \) agree, the equality also clearly holds.

Now that we have proven the existence of \( E \) let

\[
E(x) = \sum_{i=0}^{e} \gamma_i x^i
\]

\[
P_{\vec{a}}(x) = \sum_{i=0}^{k-1} a_i x^i
\]

We can now construct a system of equations by plugging in the values in \( S \).

\[
\forall x \in S, \ f(x)E(x) = P_{\vec{a}}E(x) \quad (7)
\]

Notice that this is a quadratic system of equations, and in general this is NP-hard. There is a clever way to avoid this that will result in an efficient algorithm.

**The Welch-Berlekamp Algorithm**

On input \( c' \) or \( f \), let \( N(x) = \sum_{i=1}^{e+k-1} n_i x_i \).

Let \( E(x) = \sum_{i=0}^{e} \gamma_i x_i \).

Solve for \( \{n_i, \gamma_i\} \) in the following system of equations

\[
\forall x \in S, \ f(x)E(x) = N(x) \quad (8)
\]

Output \( p = N/E \).

**Proof:** We start off by observing that by choosing \( E^* = \prod_{i=1}^{e} (x - \beta_i) \) and \( N^* = P_{\vec{a}}(x)E(x) \),
indeed \( N^*, E^* \) satisfy \( \forall x \in S, \ f(x)E^*(x) = N^*(x) \).

Thus, if we can show that for any \((N_1, E_1)\) and \((N_2, E_2)\) that satisfy

\[
f(x)E(x) = N(x)
\]

then \( N_1/E_1 = N_2/E_2 \), the correctness of the algorithm will follow.

Let \( Q \equiv N_1(x)E_2(x) - N_2(x)E_1(x) \).

\[
\forall y \in S, \ N_1(y)E_2(y) = f(y)E_1(y)E_2(y) = E_1(y)f(y)E_2(y) = E_1(y)N_2(y)
\]

All \( x \in S \) are roots of \( Q \) (recall \(|S| = n\)). Additionally, we note that \( \deg(Q) \leq 2e + k - 1 \).

Recall that the Reed Solomon code has distance \( d = n - (k - 1) \), and thus combinatorially we can only correct codes up with \( e \) less than \( \frac{d}{2} \). So \( e < \frac{n-(k-1)}{2} \), or \( 2e + k - 1 < n \).

\[
\deg(Q) \leq 2e + k - 1 < n
\]

\( Q \) has \( n \) roots but has degree less than \( n \) so \( Q \) must be the zero polynomial. This means \( N_1/E_1 = N_2/E_2 \), which finishes the proof of correctness.

### 3 Reed-Muller Codes

We now introduce a multivariate version of the Reed Solomon code.

**Definition 3.1.** The Reed-Muller code with \( m \) variables and degree \( r \), \( RS(m,r)_2 \) is constructed using a multivariate polynomial of at degree at most \( r \).

\[
P(x_1, x_2, ..., x_m) = \sum_{I \subseteq [m], |I| \leq r} a_I x^I,
\]

where \( x^I = \prod_{i \in I} x_i \) and \( a_I \in \mathbb{F}_2 \)

The number of coefficients is

\[
\sum_{i=0}^{r} \binom{m}{i} = \binom{m}{\leq r}
\]

The message is the vector of coefficients. To encode a message, we evaluate \( P \) at all points of \( \mathbb{F}_2^m \). Thus, the block length of the code is \( 2^m \). Note that the Reed-Muller code is linear.

The Reed-Muller code is a \( [2^m, (m \leq r), d] \) code, where we will see later that \( d = 2^m - r \).

**Definition 3.2.** The Hadamard Code is a special case of a Reed Muller Code, where we set \( r = 1 \).

Thus, the Hadamard Code is a \( [2^m, m, 2^{m-1}] \) code. It is not hard to see that our construction of a pairwise independent distribution (from an earlier class) is simply to output a random codeword of the Hadamard code.