AGENDA

- The story so far
- FRS codes
  - Definition
  - First Pass
  - Next Pass

THE STORY SO FAR:

- Reed-Solomon Codes! Awesome!
- List-Decoding Capacity: \( p = 1 - R \)
- Guruswami-Sudan: Can efficiently list-decode RS codes up to \( p = 1 - R \)
- Question: Can we efficiently list-decode RS codes up to \( p = 1 - R \)?
  - Answer: No or probably not, depending on the RS code.
    - If equal pts = \( F_q \)
      - Ben-Sasson, Kopparty, Radhakrishnan
      - It's as hard as discrete log
      - Cheng, Wan

Today we will see the happy ending to the story, via...

**Folded Reed-Solomon Codes.**

These were the first codes known to achieve capacity with explicit algs [Guruswami, Rudra 2008] and since then there have been several other constructions.
1. Folded Reed Solomon Codes

A Folded RS code is obtained by "folding" RS codes:

Start with an RS code with eval pts \(1, \gamma, \gamma^2, \gamma^3, \ldots, \gamma^{n-1}\), where \(n=q-1\), \(\gamma\) is a primitive element of \(\mathbb{F}_q\).

Suppose that \(m \mid n\). Then consider the following operation on a codeword from the RS code:

\[
\begin{align*}
\mathcal{F}_n & \xrightarrow{\text{Fold}} \mathcal{F}_{n/m} \\
\{ f(1), f(\gamma), f(\gamma^2), f(\gamma^3), \ldots, f(\gamma^{n-1}) \} & \xrightarrow{\text{Fold}_m \in (\mathbb{F}_q^n)^{n/m}} \mathcal{F}_{n/m} \\
c \in \mathcal{R}_q(\langle 1, \gamma, \ldots, \gamma^{n-1} \rangle, n, k) & \Rightarrow \text{Fold}_m(c) \\
\end{align*}
\]

Define the \(m\)-folding \(\text{Fold}_m(C')\) of a code \(C'\) as \(\{ \text{Fold}_m(c) : c \in C' \}\).
**Def.** Let $C' = RS_q((1, r, \ldots, r^{n-1}), n, k)$, where $n=q-1$ and $r \in \mathbb{F}_q$ is a primitive element. Let $m|n$.

The **Folded RS Code** corresponding to $C'$ is $\text{Fold}_m(C')$.

The length of $\text{Fold}_m(C')$ is $N = n/m$.

The alphabet is $\Sigma = \mathbb{F}_q^m$.

**Note:** Since folding just shuffles around the symbols, the rate does not change. Formally,

$$\text{Rate}(C) = \frac{\log_q |C|}{N} = \frac{\log |C|}{n} \cdot \frac{n}{m} \cdot \log(q^m) = \frac{\log |C|}{n \log(q)} = \frac{\log |C'|}{n} = \frac{\log_q |C'|}{n} = \text{Rate}(C')$$

Why should this be a good idea? It "restricts" the power of the adversary:

**World 1:** Bad guy gets to corrupt a $p$-fraction of an RS codeword.

$$\begin{array}{c}
 f(x) \\
 f(r) \\
 f(r^2) \\
 \vdots \\
 f(r^m) \\
 \vdots \\
 f(r^n) \\
 \end{array}$$

Here, he has much more freedom — he can corrupt any set of symbols he wants from the original codeword.

**World 2:** Bad guy gets to corrupt a $p$-fraction of a folded RS codeword.

$$\begin{array}{c}
 (f(x), f(r), f(r^2), \ldots, f(r^m)) \\
 (f(r^m), f(r^m), \ldots, f(r^n)) \\
 (f(r^n), \ldots, f(r^n)) \\
 \vdots \\
 (f(r^n), \ldots, f(r^n)) \\
 \end{array}$$

Here, much less freedom. Only certain error patterns in the original codeword are allowed.
We will prove (or at least sketch):

**THM.** Let \( 1 \leq s \leq m \). Let \( C \) be an \( m \)-folded RS code of rate \( R \). Then \( C \) is \((p, L)\)-list-decodable for

\[
p = \frac{s}{s+1} \left( 1 - \left( \frac{m}{m-s+1} \right) R \right)
\]

with list size \( L = q^s \). Moreover, the list is contained in an affine subspace of dimension \( s \).

Choose \( m = \frac{1}{\epsilon^2}, \ s = \frac{1}{\epsilon} \). Then \( L = q^{\frac{1}{\epsilon}}, \) and

\[
\frac{s}{s+1} \left( 1 - \left( \frac{m}{m-s+1} \right) R \right) = \frac{1/\epsilon}{1/\epsilon + 1} \cdot \left( 1 - \left( \frac{1/\epsilon^2}{1/\epsilon - 1/\epsilon + 1} \right) R \right) = 1 - R - O(\epsilon)
\]

So folded RS codes will give us capacity-achieving list-decodable codes:

- Rate \( R \)
- \( p = 1 - R - O(\epsilon) \) \( \rightarrow \) This trade-off is optimal.
- \( L = q^{1/\epsilon} = (N/\epsilon^2)^{1/\epsilon} \) \( \leftarrow \) This is NOT optimal... should probably be \( 1/\epsilon \).
- \( |\Sigma| = q^{1/\epsilon^2} = (N/\epsilon^2)^{1/\epsilon^2} \) \( \leftarrow \) This is also NOT optimal... we'd like it to be constant.

Subsequent work has given codes which get basically all of the desiderata, although there are still many open questions.
First Pass: Let's ignore that parameter "s."

Here is the decoding algorithm. It has a familiar outline.

Folded RS Decoder Take I.

Input: \( y = \left[ \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{m-1} \end{pmatrix}, \begin{pmatrix} y_m \\ y_{m+1} \\ \vdots \\ y_{2m-1} \end{pmatrix}, \ldots, \begin{pmatrix} y_{n-m} \\ y_{n-m+1} \\ \vdots \\ y_{n-1} \end{pmatrix} \right] \in \left( F_q^m \right)^N \\

as well as an agreement parameter \( t \), and a parameter \( D \), and \( k \leq n \).

Output: A list \( L \) containing all polynomials \( f(x) \in F_q [x] \) of degree \( < k \), so that for at least \( t \) values of \( i \in [0, N-1] \),

\[
\begin{pmatrix}
(f(y^{mi})) \\
\vdots \\
f(y^{mi+m-1})
\end{pmatrix}
= 
\begin{pmatrix}
y_{mi} \\
\vdots \\
y_{mi+m-1}
\end{pmatrix}
\]

STEP 1. (Interpolation)

Find a nonzero polynomial

\[
Q(x, Y_1, Y_2, \ldots, Y_m) = A_0(x) + A_1(x) \cdot Y_1 + \ldots + A_m(x) \cdot Y_m
\]

with \( \deg(A_0) \leq D + k - 1 \), and \( \deg(A_i) \leq D \) for \( i = 1, \ldots, m \),

so that

\[
Q(y^{mi}, y_{mi}, y_{mi+1}, \ldots, y_{mi+m-1}) = 0 \quad \forall \ i \in \{0, \ldots, N-1\}
\]

\( X \) takes on the "first" eval pt for that symbol

\( Y_1, \ldots, Y_m \) take on the symbol \( \begin{pmatrix} y_{mi} \\ \vdots \\ y_{mi+m-1} \end{pmatrix} \)

ctd...
Folded RS decoder Takei continued...

STEP 2. (Root-Finding)

Find all the polys \( f \) so that

\[
Q(X, f(X), f(\gamma X), \ldots, f(\gamma^{m-1} X)) = 0,
\]

and return them.

This alg is not our final algorithm, and it only gets

However, the main ideas will come through by analyzing this (easier) version, so let's start there.

As usual, we need to do three things.

1. Set parameters
2. Show that we can find such a \( Q \) ← As usual, via linear algebra.
3. Show that STEP 2 is a good idea. ← As usual, low-deg polys don't have many roots.
4. Show that we can do STEP 2 efficiently. ← Usually this one follows since we can factor polys, but now we'll switch it up a bit.

\[ p = \left( \frac{m}{m+1} \right) (1-m \cdot R). \]

\[
D = \left\lfloor \frac{N-k+1}{m+1} \right\rfloor
\]

\[ D+k+m(D+1) = (m+1)(D+1) + k-1 \text{ coefficients}. \]

\[
\text{Coefs of } X^j \text{ that show up in } A_0 \quad \text{Coefs of } \gamma_i \cdot X^j \text{ that show up in each } A_i
\]

Our choice of \( N \) ensures that this is

\[ \#\text{coeffs} > N = \#\text{constraints}, \]

so we can find the desired \( Q \).
Let $R(X) = \mathcal{Q}(X, f(x), f(\gamma X), \ldots, f(\gamma^n X))$

$$= A_0(X) + \frac{f(x) \cdot A_1(X)}{\deg D+k-1} + \cdots + \frac{f(\gamma^n X) \cdot A_m(X)}{\deg D+k-1}$$

So $\deg(R) \leq D+k-1$. 

Suppose that $\text{Fold}_m(f)$ agrees with $y$ in at least $t$ places. Then $R$ has at least $t$ roots, so $R \equiv 0$ as long as

$$\left(\#\text{roots}\right) > \deg(R), \quad t > D+k-1 = \left\lfloor \frac{N-k+1}{m+1} \right\rfloor + k-1.$$ 

So it would be enough if

$$t \geq \frac{N-k}{m+1} + \frac{1}{m+1}$$

$$= \frac{N}{m+1} + \frac{m-1}{m+1}$$

$$= N \left(\frac{1}{m+1} + \frac{m-1}{m+1}\right),$$

which is what we chose.

Therefore, if $f$ agrees with $y$ too much, we will return it.

iv. How do we actually find this list? Is it small?

**PUNT.** (we'll come back to this).
Fixing up the first pass.

We will tweak things to get **more roots** in our interpolating poly.

Suppose that

\[
\begin{pmatrix}
  f(y_{mi}) \\
  \vdots \\
  f(y_{mi+m-1})
\end{pmatrix}
= 
\begin{pmatrix}
  y_{mi} \\
  \vdots \\
  y_{mi+m-1}
\end{pmatrix}
\]

We were using the root of \( Q \) at

\[
Q(y_{mi}, f(y_{mi}), \ldots, f(y_{mi+s-1}), f(y_{mi+s}), \ldots, f(y_{mi+m-1})) = 0.
\]

But we could get **more roots** if we did something like this:

\[
Q(y_{mi}, f(y_{mi}), \ldots, f(y_{mi+s-1})) = 0
\]

\[
Q(y_{mi+s}, f(y_{mi+s}), \ldots, f(y_{mi+m-1})) = 0
\]

\[
\vdots
\]

\[
Q(y_{mi+m-s}, f(y_{mi+m-s}), \ldots, f(y_{mi+m-1})) = 0
\]

Basically, we take a sliding window across each symbol and turn that into a constraint.

There's a trade-off here: **more agreement**! **more constraints**.

\[ \leftarrow \text{So we get better "e" for a given "D."} \]

\[ \leftarrow \text{So } D \text{ has to be smaller in order to be able to find } Q. \]

But this trade-off turns out to be beneficial!
The changes from TAKE I are highlighted in RED.

Input: \( y = \begin{bmatrix} (y^0_0, \ldots, y^0_m) \\ (y^1_0, \ldots, y^1_m) \\ \vdots \\ (y^{n-m}_0, \ldots, y^{n-m}_m) \end{bmatrix} \in (F_q^m)^N \)

as well as an agreement parameter \( t \), and a parameter \( D \), and \( k \leq n \).

Output: A list \( L \) containing all polynomials \( f(X) \in F_q[X] \) of degree \( < k \), so that for at least \( t \) values of \( i \in [0, N-1] \),

\[
\begin{bmatrix} f(y^i_m) \\ \vdots \\ f(y^{i+m-1}_m) \end{bmatrix} = \begin{bmatrix} y^i_m \\ \vdots \\ y^{i+m-1}_m \end{bmatrix}
\]

STEP 1. (Interpolation)

Find a nonzero polynomial

\[
Q(X, Y_1, Y_2, \ldots, Y_s) = A_0(X) + A_1(X) \cdot Y_1 + \ldots + A_s(X) \cdot Y_s
\]

with \( \deg(A_0) \leq D + k - 1 \), and \( \deg(A_i) \leq D \) for \( i = 1, \ldots, s \)

so that

\[
Q(x^im+j, y^im+j, \ldots, y^{im+j+s-1}) = 0 \quad \forall 0 \leq i < N \\
A 0 \leq j \leq m-s.
\]

STEP 2. (Root-finding)

Find all the polys \( f \) so that

\[
Q(X, f(X), f(yX), \ldots, f(y^{s-1}X)) = 0,
\]

and return them.
Again, we go through our steps i, ii, iii, iv.

i. SET PARAMETERS: Suppose

\[ D = \left\lfloor \frac{N(m-s+1) - k + 1}{s+1} \right\rfloor, \quad t > \left\lfloor \frac{D + k - 1}{m - s + 1} \right\rfloor \]

ii. We can find \( Q \). **FUN EXERCISE!** (exactly the same as usual)

iii. STEP 2 is a good idea.

**OUTLINE:**

\begin{align*}
\text{\{FUN EXERCISE! Fill in the details.\}} & \\
\text{\{FUN EXERCISE! Fill in the details.\}} & \\
\text{\{FUN EXERCISE! Fill in the details.\}} & \\
\text{\{FUN EXERCISE! Fill in the details.\}} & \\
\end{align*}

\[ R(X) := Q(X, f(\gamma X), ..., f(\gamma^{s-1} X)) \]

has at least \( t(m-s+1) \) roots.

- The degree of \( R(X) \) is \(< t(m-s+1) \) with our choices of parameters.

iv. We can efficiently find all polys \( f \) so that \( Q(X, f(X), ..., f(\gamma^{s-1} X)) \equiv 0 \).

The way we would normally do this is argue that since the degree of \( Q \) is small there can't be too many \( f \)'s.

That breaks down a bit here since each instance of \( f \) is a little different.

Instead, we'll go back to **LINEAR ALGEBRA**.

Here, we will just sketch how this goes.

See Chapter 14 of **ESSENTIAL CODING THEORY** for details.
We want to find all $f$ so that

$$(*) \quad R(X) := Q(X, f(X), f(\gamma X), \ldots, f(\gamma^{s-1} X)) \equiv 0.$$ 

Say that $f(X) = f_0 + f_1 X + f_2 X^2 + \ldots + f_{k-1} X^{k-1}$.

We can write $(*)$ as a giant linear equation in the coefficients $f_0, f_1, \ldots, f_{k-1}$.

**Question:** What is the constant term in $R(X)$? That is, if $R(X) = \sum_j r_j X^j$, what is $r_0$?

**Answer 1:** $r_0 = 0$, since $R \equiv 0$.

**Answer 2:** Okay, let's compute it. Recall that

$$R(X) = Q(X, f(X), \ldots, f(\gamma^{s-1} X)) = A_0(X) + f(X) \cdot A_1(X) + \ldots + f(\gamma^{s-1} X) \cdot A_s(X),$$

and so

$$r_0 = R(0) \quad \text{[Here, } A_i(X) = \sum_j a_{ij} X^j \text{]}$$

$$= A_0(0) + f(0) \cdot A_1(0) + \ldots + f(0) \cdot A_s(0)$$

$$= a_{00} + \sum_{j=1}^s a_{j0} \cdot f_0$$

$$= a_{00} + f_0 \left( \sum_{j=1}^s a_{j0} \right)$$

So actually we know $f_0$! $f_0 = \left( \frac{-a_{00}}{\sum_{j=1}^s a_{j0}} \right)$. 
It turns out that we can keep doing this, and moreover it turns out that the
linear system that we get is triangular:

\[ M = \begin{bmatrix} f_0 & f_1 & \cdots & f_{k-1} \\ \end{bmatrix} \]

Stuff we know from Q

This entry is \( \sum_j a_{j0} \)

This entry is \( -a_{00} \)

Even better, we can exactly figure out what goes on the diagonal:

\[ M = \begin{bmatrix} B(0) & B(1) & \cdots & B(k-1) \\ \end{bmatrix} \]

\[ \text{Where } B(X) = a_{10} + a_{20}X + a_{30}X^2 + \cdots + a_{s,0}X^{s-1} \]

\[ \text{B has at most } s-1 \text{ roots} \]
\[ \Rightarrow M \text{ has at most } s-1 \text{ 0's on the diagonal} \]
\[ \Rightarrow \dim(Ker(M)) \leq s-1 \]
\[ \Rightarrow \text{There are at most } q^{s-1} \text{ solutions to this linear system.} \]

That's what we wanted!

So, modulo the details, we've proved the **ThM** from the beginning of the lecture.

Check out **Essential Coding Theory**, Chapter 14, for the details!
QUESTIONS to PONDER

① Work out the details for part (iv)
② Can you get a smaller list size for FRS codes?
③ Can you extend this algorithm to do list recovery?