# Class 5 Exercises

CS250/EE387, Winter 2025

In this class, we'll investigate/develop the (starting point[1] for the) *Berlekamp-Massey Algorithm* for decoding Reed-Solomon codes. Some notation:

- We will be working with a RS code $C \subseteq \mathbb{F}_q^n$ with length $n = q - 1$ over $\mathbb{F}_q$ and with evaluation points $1, \gamma, \gamma^2, \ldots, \gamma^{n-1}$.
- We will try to decode $C$ from $e$ errors. Suppose that $v \in \mathbb{F}_q^n$ is the received word, so $v = c + p$ for $c \in C$ and $p \in \mathbb{F}_q^n$ is an error vector so that $\mathrm{wt}(p) \leq e$.
- Let $p = (p_0, \ldots, p_{n-1})$ be the error vector, and let $E = \{i : p_i \neq 0\}$. be the locations of the errors.
- Let $d = n - k + 1$ be the distance of the RS code, and assume that $e \leq \left\lfloor \frac{d-1}{2} \right\rfloor$, so that unique decoding is possible.

Now onto the questions.

1. The first step of the Berlekamp-Massey algorithm is to compute the *syndrome* $s = Hv$, where $H$ is the parity-check matrix for $C$. Write $s = (s_1, \ldots, s_{d-1})$, and let $s(Z) = \sum_{i=1}^{d-1} s_i Z^i$.

   Show that
   $$s(Z) = \sum_{i \in E} p_i \cdot \left( \frac{\gamma^i Z - (\gamma^i Z)^d}{1 - \gamma^i Z} \right).$$

   *Hint/Outline:*

   - *First, show that* $s(Z) = \sum_{i \in E} p_i \sum_{\ell=1}^{d-1} (\gamma^i Z)^\ell$. *(To do that, write out what* $s(Z)$ *is, in terms of the* $\gamma^i$*'s and* $p_j$*'s, by using the definition of the parity-check matrix and the fact that* $Hv = s$*).*
   - *Then, remember that* $\sum_{j=1}^{n} x^j = \frac{x^{n+1} - x}{x - 1}$...

2. Let $\sigma(Z) = \prod_{i \in E} (1 - \gamma^i Z)$. Explain why we will be in good shape for decoding if we can figure out what $\sigma$ is.

3. Consider $\sigma(Z) \cdot s(Z)$. Show that this can be written as
   $$\sigma(Z) \cdot s(Z) = w(Z) + Z^d r(Z),$$
   where $w(Z)$ and $r(Z)$ are polynomials, and $\deg(w) \leq e$. Write down an expression for $w(Z)$.

4. Let's explore this polynomial $w(Z)$ a bit more.

   (a) For all $r \in E$, $w(\gamma^{-r}) = p_r \cdot \prod_{j \in E \setminus \{r\}} (1 - \gamma^{j-r})$. Use this to explain why we will be done (that is, we can figure out the error vector $p$) if we can figure out *both* $\sigma(Z)$ and $w(Z)$.

[1] In fact, what's usually called the "Berlekamp-Massey Algorithm" picks up where these questions leave off. Today we are going to design an algorithm that runs decently fast. The BM algorithm starts with this outline and makes it much faster.

(b) **(Optional:)** $w(Z)$ and $\sigma(Z)$ are relatively prime. That is, there is no polynomial $g(Z)$ (other than $g(Z) \equiv 1$) that divides them both.

*Note: if you don't feel like showing this, take it as given and skip this part; you can come back and think about it later if you have time!*

5. The previous part implies that, for all $r$ with $e+1 \leq r \leq d-1$, we have

$$\text{coefficient on } Z^r \text{ in } s(Z)\sigma(Z) = 0.$$

What is that coefficient, in terms of the coefficients of $s$ (which we know) and the coefficients of $\sigma$ (which we don't know)? Write down a system of $d-e-1$ linear constraints that the coefficients of $\sigma$ must satisfy. Your constraints should be in terms of the $s_i$. Explain why there is at least one solution to this system of equations.

6. Suppose we were to solve your system of equations to obtain $(\tilde{\sigma}_0, \ldots, \tilde{\sigma}_{n-1})$ and a corresponding polynomial $\tilde{\sigma}(Z) = \sum_{i=0}^{e} \tilde{\sigma}_i Z^i$. Explain why we can write

$$s(Z)\tilde{\sigma}(Z) = \tilde{w}(Z) + Z^d \tilde{r}(Z)$$

for some polynomials $\tilde{w}(Z), \tilde{r}(Z)$ with $\deg(\tilde{w}) \leq e$.

*Hint: Don't overthink it. (Think about what we did to answer the previous question...)*

7. Show that $\tilde{\sigma}(Z)w(Z) = \sigma(Z)\tilde{w}(Z)$.

*Hint: Consider $\sigma(Z)s(Z)\tilde{\sigma}(Z)$. Further hint: $(\sigma(Z)s(Z))\tilde{\sigma}(Z) = \sigma(Z)(s(Z)\tilde{\sigma}(Z))$.*

8. Explain how, given $\tilde{\sigma}$ and $\tilde{w}$, to find $\sigma$ and $w$.

*Hint: By Question 7, $\frac{\tilde{w}(Z)}{\tilde{\sigma}(Z)} = \frac{w(Z)}{\sigma(Z)}$.... Part 4(b) might be useful. Further hint: Suppose that I was thinking of two integers that are relatively prime (like 2 and 5). Suppose I told you that the one divided by the other was the same as 40/100. How could you recover the two numbers?*

9. Put all the pieces together to write down an efficient (polynomial-time) algorithm to recover the codeword $c$ given $v$. What is the running time of your algorithm, in terms of the number of operations over $\mathbb{F}_q$?

If it helps, finding the gcd (or reducing a fraction) of two degree-$D$ polynomials (with, say, Euclid's algorithm) takes $O(D^2)$ operations over $\mathbb{F}_q$. Finding the roots of a degree-$D$ polynomial in $\mathbb{F}_q$ can be done with $O(D^2 \log q)$ operations over $\mathbb{F}_q$. Evaluating a degree $D$ polynomial at a point can be done with about $O(D \log D)$ operations over $\mathbb{F}_q$.

10. **(Bonus)** Can you think of a way to speed up your algorithm? What steps seem especially lossy?

11. **(Bonus)** A *linear feedback shift register* (LFSR) is defined as follows. We have a *register*, which initially holds $t$ field symbols $s_0, s_1, \ldots, s_{t-1} \in \mathbb{F}_q$. Fix some vector $\vec{\tau} \in \mathbb{F}_q^t$. We are going to modify this register over time, as follows. At time $j$, suppose that the register holds $(s_j, s_{j+1}, \ldots, s_{j+t-1})$. To move to time $j+1$, we do the following:

- Write the contents of the register as $\vec{s} = (s_j, \ldots, s_{j+t-1})$, and compute $s_{j+t} = \vec{s} \cdot \vec{\tau}$.
- Shift all the contents of the register over by one, dropping the first one, so now it looks like $(s_{j+1}, \ldots, s_{j+t-1}, \text{----})$.
- Fill in the last black with $s_{j+t}$, so now we have $(s_{j+1}, \ldots, s_{j+t})$.

Then we repeat for a while! A question that one might ask about such a thing is: *Given a sequence of "dropped" bits $s_0, s_1, s_2, \ldots, s_N$, can you recover the coefficient vector $\vec{\tau}$?*

If you look up the *Berlekamp-Massey algorithm* on the internet, it is often described as a very fast way to solve this LFSR problem. But today we said that it was a fast algorithm for decoding RS codes. What does the LFSR problem have to do with decoding Reed-Solomon codes??