An Introduction to Reed-Solomon Codes

Handwritten Notes by Jack Keil Wolf (1998)
Typeset by Paula Evans (2006)
Expanded by Henry D. Pfister (2006-)
June 4th, 2021

1   Polynomial Fitting and Reed-Solomon Codes

1.1   Basic Idea

An introduction to Reed-Solomon codes will be presented that requires no knowledge of coding theory. It relies on a very simple method of "fitting" points in a plane by the polynomial of smallest degree that passes through those points. For example, we know that any two points in the plane can be "fitted" uniquely by a straight line, any 3 points can be fitted uniquely by a parabola, and so on. This is illustrated in Figure 1. In general, we find that k points can be "fitted" by polynomial of the form

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

The formula for fitting $y = f(x)$ is not important here, but it is called the Lagrange Interpolation formula. Instead, we can start with a polynomial of degree $k - 1$,

$$y = f(x) = f_{k-1}x^{k-2} + \ldots + f_2 x^2 + f_1 x + f_0,$$

and evaluate it at n (where $n \geq k$) *distinct* values of x, say $x_0, x_1, \ldots, x_{n-1}$ to obtain the n points in the plane $(y_0, x_0), \ldots, (y_{n-1}, x_{n-1})$ where $y_i = f(x_i)$.

From the Lagrange Interpolation formula, we know that the polynomial $f(x)$ can be obtained from any k of the n points (since the polynomial $f(x)$ is of degree $k-1$). If $n = k$ we have just enough points. If $n > k$, we have more than enough points. The $(n-k)$ extra points are "redundant" and can help in choosing the correct $f(x)$ even if some of the points have been "moved".

We illustrate this by the following example. Assume that $k = 3$ and $n = 7$ so that the 7 points are initially on a parabola $f(x) = f_2 x^2 + f_1 x + f_0$ as shown in Figur 1c.

Assume that for some mysterious reason 2 of the 7 points have been moved *vertically* resulting in the 7 points shown in Figure 1d (vertical movement means that the x component has not changed). It is clear that the 7 points shown in Figure 1d cannot be fitted by a single parabola. *Our problem is to find the single parabola that fits as many as possible of the 7 points.* In statistics, one would treat the moved points as outliers. It is our claim that in this case, the original parabola is the parabola that fits the maximum number of points. We can see this by noting that:

- The original parabola fits the $(7 - 2) = 5$ unmoved points

- If a parabola fits 5 points, then at least $5 - 2 = 3$ of them are unmoved it must equal the original parabola. Thus, an incorrect parabola fits at most 4 points

(a) Line   (b) Parabola   (c) Oversampled parabola   (d) With 2 points moved
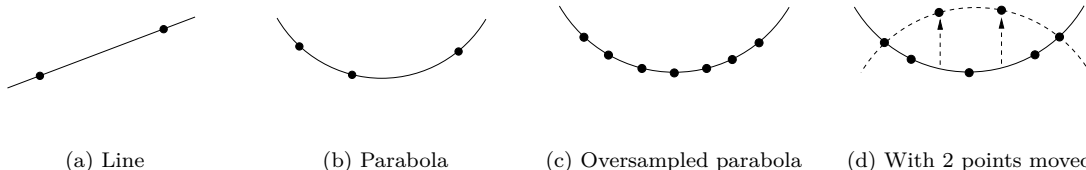
Figure 1: Fitting points on a plane and example with $n = 7$ and $k = 3$.

The first statement is obvious since the original parabola was used to find the original points so certainly will fit the 5 unmoved points. The validity of the second statement takes some explanation. As shown in Figure 1d we have attempted to move the 2 points vertically in such a way so that as many points as possible land on another parabola (shown as a dotted curve). But, we can make only 4 points (2 moved points and 2 unmoved points) fall on this wrong parabola. Since, if there were more than 4 points on this wrong parabola, 3 of them would be unmoved points and we know that any 3 unmoved points uniquely define the original parabola.

Let us generalize this result to the case of an arbitrary choice for k and $n > k$. Assume that we start with an original polynomial of degree $k - 1$ for which we evaluate $f(x)$ at n distinct values of x. Let us take the resulting n points and move t of them (vertically). Let us now find the polynomial of degree $k-1$ that fits as many of these points as possible. I claim that the original polynomial will be the winner provided that $t \leq (n - k)/2$. To see this note that

- The original polynomial will fit the $(n - t)$ unmoved points

- Any other polynomial fits at most $(t + k - 1)$ points

  - If one fits $t + k$ points, then it fits at least k unmoved points and must equal the original

- Successful decoding guaranteed if $(n - t) \geq (t + k - 1) + 1 \Leftrightarrow t \leq (n - k)/2$.

In summary, we find that, if there are $(n - k)$ redundant points, then we can recover the original $f(x)$ as long as the number of errors is less than half this number.

Before examining how the above result relates to Reed-Solomon codes, let us consider what type of number system applies to the previously described system. The reader has most likely assumed that x and y are real variables ($\infty < x < \infty, -\infty < y < \infty$) and that the coefficients of $f(x)$ are themselves real numbers. This is the usual case when one uses Lagrange Interpolation. However we can consider that x, y take values from a finite field: a finite set of values that can be added, subtracted, multiplied and divided in a manner analogous to real numbers. One such set of values are the 2^m binary vectors of dimension m. In particular we can start with the polynomial $f(x) = f_{k-1}x^{k-1} + f_{k-2}x^{k-2} + \cdots + f_2x^2 + f_1x + f_0$ where the coefficients $\{f_i\}_{i=0}^{k-1}$ are binary vectors of dimension m, and x is itself constrained to be one of those binary vectors. When we substitute for x one of the binary m-vectors, the resultant value of y is also one of these binary m-vectors. Everything we developed before still hold except now we see that n must be less than or equal to 2^m, since there are at most 2^m distinct values that can be substituted for x in $y = f(x)$. Specifically the m points: $(y_0, x_0), \ldots, (y_{n-1}, x_{n-1})$ as such that the x and y component are binary m vectors. Still, we can have up to $(n - k)/2$ of these points move vertically (i.e, change their value of y) and still recover the original $f(x)$.

1.2   Reed-Solomon Codes

We now arrive at the relationship between fitting polynomials to curves and Reed-Solomon codes. First, we must define Reed-Solomon codes properly. For our present purposes, a Reed-Solomon code is a set of length-n vectors (known as codewords), where the elements of the vector (known as symbols) consist of m binary digits. Our only restriction is that n must be chosen no larger than 2^m. Of the n symbols in each code word, k of them carry information and the other $(n - k)$ are redundant symbols. At the transmitter side, an *encoder* is used to compute the $(n - k)$ redundant symbols from the k information symbols. Then the n symbols of a code word are transmitted (or stored). When the n symbols are received (or read), some of them have errors. Note that, since a symbol is a binary m-vector, we say that a symbol is in error if *one or more* of the m binary digits are incorrect.

Assume that, of the total n symbols, exactly t of them are received in error (and the other $n - t$ are received correctly). Reed-Solomon codes have the remarkable property that if $t \leq (n - k)/2$, then the correct information can be computed from this faulty codeword. Furthermore, if s of the received symbols are *erased* (i.e, tagged as probably being faulty) *and* another t symbols are received in error, the correct information can be computed from the faulty code word provided that $s + 2t \leq n - k$. The device that reconstructs the information from the received vector is called a decoder.

We are now ready to relate the Reed-Solomon code to Lagrange Interpolation. As you might suspect, we will use points with x and y components being binary m-vectors. We assume that the k information symbols correspond to the y vectors of the points $(y_{i_0}, x_{i_0}), (y_{i_1}, x_{i_1}), \ldots, (y_{i_{k-1}}, x_{i_{k-1}})$ where

$x_{i_0}, x_{i_1}, x_{i_2}, \ldots, x_{i_{k-1}}$ are any k distinct x-values. Remember that there are a total of 2^m distinct values to choose from. From these k points, we compute the unique polynomial of the form $f(x) = f_{k-1}x^{k-1} + f_{k-2}x^{k-2} + \ldots + f_2x^2 + f_1x + f_0$ which fits these points. To encode the k information symbols to a codeword of length m symbols, we evaluate $f(x)$ at $(n-k)$ additional points. The codeword then consists of the original k information symbols $y_{i_0}, y_{i_1}, y_{i_2}, \ldots y_{i_{k-1}}$ plus the y values of the $n-k$ additional points. To keep the nomenclature simple, we will assume that a codeword is of the form: $(y_0, y_1, \ldots, y_{n-1})$ where $(y_0, y_1, \ldots, y_{k-1})$ are the information symbols and $(y_k, y_{k+1}, \ldots, y_{n-1})$ are the redundant symbols. Thus, we have shown how to encode.

Suppose we receive the noisy code word $(\widetilde{y}_0, \widetilde{y}_1, \ldots, \widetilde{y}_{n-1})$ where $\widetilde{y}_i = y_i$ for $(n-t)$ of the symbols (i.e. the symbols without error) and $\widetilde{y}_i \neq y_i$ for t of the symbols (i.e., those symbols received in error). We now take the received code word (with errors) and treat them as n points in the plane. We note that the x values are known to the decoder and are the same values used in the encoder . We now find the polynomial $f(x)$ of degree $k-1$ or less which fits the maximum number of points. We know from our earlier work that this will be the correct $f(x)$ provided that $t \leq (n-k)/2$. Once $f(x)$ is found, we can evaluate it at the x values corresponding to the information positions to get the correct values for all of the information symbols. Furthermore, if there are s erasures, the only effect is to reduce n by s so that we will get correct decoding if $t \leq (n-s-k)/2$.

2 Evaluation Points and the Fourier Transform

One of the easiest ways to understand the algebraic decoding of Reed-Solomon codes is to choose a special set of evaluation points over the complex numbers. If we let the n evaluation points be $x_j = e^{2\pi ij/n}$ (where $i = \sqrt{-1}$), then the polynomial evaluation

$$
\begin{aligned}
f(x_j) &= f_{k-1}x_j^{k-1} + f_{k-2}x_j^{k-2} + \ldots + f_2x_j^2 + f_1x_j + f_0 \\
&= \sum_{l=0}^{k-1} f_l e^{2\pi ijl/n}
\end{aligned}
$$

is proportional to the inverse discrete Fourier transform of the n-point zero padded sequence $(f_0, f_1, \ldots, f_{k-1}, 0, \ldots, 0)$. So the codeword is defined by $y_j = f(x_j)$ and the noisy received codeword is $\widetilde{y}_j = y_j + z_j$ where z_j is noise contribution to the jth point. If we form the received polynomial

$$
\widetilde{f}(x) = \widetilde{y}_{n-1}x^{n-1} + \widetilde{y}_{n-2}x^{n-2} + \ldots + \widetilde{y}_1x^1 + \widetilde{y}_0,
$$

then we find that

$$
\begin{aligned}
\widetilde{f}(x_j^{-1}) &= \sum_{l=0}^{n-1} \widetilde{y}_l e^{-2\pi ijl/n} \\
&= \sum_{l=0}^{n-1} y_l e^{-2\pi ijl/n} + \sum_{l=0}^{n-1} z_l e^{-2\pi ijl/n}.
\end{aligned}
$$

Notice that the first sum is the discrete Fourier transform of $(y_0, y_1, \ldots, y_{n-1})$ which is zero for $j = k, k+1, \ldots, n-1$. So, $\widetilde{f}(x_j^{-1})$ depends only on the noise for $j = k, k+1, \ldots, n-1$. Assuming that exactly t elements of the noise $(z_{\sigma(1)}, z_{\sigma(2)}, \ldots, z_{\sigma(t)})$ are non-zero, this gives, for $j = 1, 2, \ldots n-k$,

$$
\begin{aligned}
\widetilde{f}(x_{n-j}^{-1}) &= \sum_{l=0}^{n-1} z_l e^{-2\pi i(n-j)l/n} \\
&= \sum_{l=0}^{n-1} z_l e^{2\pi ijl/n} \\
&= \sum_{l=1}^{t} z_{\sigma(l)} \left(e^{2\pi i\sigma(l)/n}\right)^j.
\end{aligned}
$$

This can be extended to a semi-infinite sequence $(S_1, S_2, \ldots)$ by defining

$$
S_j \triangleq \sum_{l=1}^{t} z_{\sigma(l)} \left(e^{2\pi i\sigma(l)/n}\right)^j,
$$

and noting that $S_j = \widetilde{f}\left(x_{n-j}^{-1}\right)$ for $j = 1, 2, \ldots, n - k$. In coding theory, this sequence is called the *syndrome sequence*. Notice that the sequence $(S_1, S_2, \ldots)$ consists of equally spaced samples of the sum of t complex exponentials with unknown frequency and amplitude. If we can solve this spectral estimation problem (i.e., find the amplitude and frequency) using the first $n - k$ samples, then we can reconstruct the z's and subtract the noise from our original signal.

Fortunately, this problem was solved in 1795 by Baron Gaspard Riche de Prony. Prony's method[1] allows one to find the frequency and amplitude of ν unknown complex exponentials using 2ν equally spaced samples. It can be viewed as a filter design problem; the goal is to find the $\nu + 1$ tap FIR filter, with transfer function $\Lambda(x) \triangleq \sum_{j=0}^{\nu} \Lambda_j x^j$, that has zeros at each of the frequencies in the input sequence $(S_1, S_2, \ldots)$, with transform

$$
\begin{aligned}
S(x) &\triangleq \sum_{j=1}^{\infty} S_j x^j \\
&= \sum_{l=1}^{t} z_{\sigma(l)} \sum_{j=1}^{\infty} x^j \left(e^{2\pi i \sigma(l)/n}\right)^j \\
&= \sum_{l=1}^{t} z_{\sigma(l)} \frac{x e^{2\pi i \sigma(l)/n}}{1 - x e^{2\pi i \sigma(l)/n}}.
\end{aligned}
$$

In this case, the filter will null the entire tail of this infinite input sequence. In coding theory, the polynomial $\Lambda(x)$ is called the *error-locator polynomial* because (as we will soon see) $\Lambda\left(e^{-2\pi i \sigma(l)/n}\right) = 0$ for $l = 1, 2, \ldots t$, so that factoring $\Lambda(x)$ allows one to find the error locations.

This means that the output sequence, with transfer function $\Omega(x) \triangleq \Lambda(x) S(x)$, will satisfy $\Omega_j = 0$ for $j = \nu + 1, \nu + 2, \ldots$ and $\Omega(x) \triangleq \sum_{j=0}^{\infty} \Omega_j x^j$. Each of these constraints gives a linear equation of the form

$$
\sum_{m=0}^{\nu} S_{j-m} \Lambda_m = 0.
$$

Assuming $\Lambda_0 = 1$ and putting these equations in matrix form for $j = \nu + 1, \nu + 2, \ldots, 2\nu$ gives

$$
\begin{bmatrix}
S_1 & S_2 & \cdots & S_\nu \\
S_2 & S_3 & \cdots & S_{\nu+1} \\
\vdots & \vdots & \ddots & \vdots \\
S_\nu & S_{\nu+1} & \cdots & S_{2\nu-1}
\end{bmatrix}
\begin{bmatrix}
\Lambda_\nu \\
\Lambda_{\nu-1} \\
\vdots \\
\Lambda_1
\end{bmatrix}
=
\begin{bmatrix}
-S_{\nu+1} \\
-S_{\nu+2} \\
\vdots \\
-S_{2\nu}
\end{bmatrix}. \tag{1}
$$

Since a $\nu + 1$ tap filter can null exactly ν distinct frequencies, this system of equations will always have a unique solution $\nu = t$. If $\nu > t$, then the filter has an extra zero which can be placed anywhere and the system of equations is underdetermined (i.e., has multiple solutions).

The desired filter for $\nu = t$ is given by

$$
\Lambda(x) = \prod_{m=1}^{t} \left(1 - x e^{2\pi i \sigma(m)/n}\right)
$$

and one can verify that, for this choice,

$$
\begin{aligned}
\Omega(x) &= \prod_{m=1}^{t} \left(1 - x e^{2\pi i \sigma(m)/n}\right) \sum_{l=1}^{t} z_{\sigma(l)} \frac{x e^{2\pi i \sigma(l)/n}}{1 - x e^{2\pi i \sigma(l)/n}} \\
&= \sum_{l=1}^{t} z_{\sigma(l)} x e^{2\pi i \sigma(l)/n} \prod_{m \neq l}^{t} \left(1 - x e^{2\pi i \sigma(m)/n}\right).
\end{aligned}
$$

---

[1]Today, Prony's method typically refers to an slightly different approach that uses more than $2t$ samples to achieve robustness in the presence of noise.

In coding theory, $\Omega(x)$ is called the *error-evaluator polynomial* because

$$\Omega\left(e^{-2\pi i \sigma(l)/n}\right) = z_{\sigma(l)} \left[x e^{2\pi i \sigma(l)/n} \prod_{m \neq l}^{t} \left(1 - x e^{2\pi i \sigma(m)/n}\right) \right]_{x = e^{-2\pi i \sigma(l)/n}}$$

$$= -z_{\sigma(l)} e^{-2\pi i \sigma(l)/n} \Lambda'\left(e^{-2\pi i \sigma(l)/n}\right),$$

where

$$\Lambda'(x) = -\sum_{l=1}^{t} e^{2\pi i \sigma(l)/n} \prod_{m \neq l}^{t} \left(1 - x e^{2\pi i \sigma(m)/n}\right).$$

The basic method for decoding Reed-Solomon codes proceeds by applying this method, for $j = 0, 1, \ldots, \frac{n-k-2}{2}$ and $\nu = \frac{n-k-2j}{2}$, to the sequence $S_1, S_2, \ldots, S_{2\nu}$ until the matrix equation (1) has a unique solution. Then, the error-locator polynomial is factored and the error-magnitudes are found. This approach to decoding was first applied to binary BCH codes by Peterson in 1960 and generalized to non-binary BCH and Reed-Solomon codes by Gorenstein and Zierler in 1961. It is now known as Peterson-Gorenstein-Zierler decoding. The connection to Prony's method was observed in 1967 by Wolf.