Decoder for Binary BCH Codes
EECS 869: Error Control Coding
Fall 2009

Complete the following tasks. You should submit an e-mail with one .m file attachment (use esp@eecs.ku.edu).

1. **Implement Berlekamp's Algorithm for Decoding Binary BCH Codes.** Attached to the end of this write-up, you will find a few pages copied from Wicker's book that contain the specification for this algorithm. You should implement this as a MATLAB function with a similar syntax to the previous project:

   ```
   c_hat = BchDecoderBerlekampXXX(r,t);
   ```

   where $\mathbf{r}$ and $\hat{\mathbf{c}}$ are $1 \times n$ MATLAB vectors containing ones and zeros and t is the error correcting strength of the code. I expect you to submit your own original MATLAB code. You should submit your function via e-mail (use esp@eecs.ku.edu), and don't forget to replace `XXX` with your first name.

Here are some tips and tricks from my own implementation:

- We will assume the codeword length is $n = 2^m - 1$ and the decoder operates over $GF(2^m)$.
- At the very beginning of your MATLAB function, you can "type cast" `r` as a Galois field object with the command `r_gf = gf(r,m)`. Once you do that, all of the MATLAB operations you perform on `r_gf` will automatically be performed over $GF(2^m)$. (By the way, `r_gf` is just a name, you don't have to have the `_gf`, this is just my style.)
- To get the primitive element α, you can just ask for the Galois field element corresponding to the polynomial $\alpha + 0$, which has a binary representation of `10` and an integer representation of 2, hence `alph = gf(2,m)` (I like to avoid using the MATLAB name `alpha` because there is a built-in function with that name).
- You will find Moon's Table 5.1 (p. 199) helpful in understanding Galois fields. Each element in the field has a unique integer assigned to it, in the range $0, 1, \cdots, 2^m - 1$ [Moon calls this the "Vector Representation (integer)"]. MATLAB uses this integer representation, and thus the integer 2 corresponds with α. In addition to the integer representation, you will also find the "Logarithm" representation helpful. You can see the logarithms of each of the elements in Moon's table, which are integers in the range $0, 1, \cdots, 2^m - 2$ and the notable case that 0 has an undefined logarithm. Lucky for us, MATLAB has "overridden" a lot of its built-in functions to work properly with Galois field elements. For example, if you have `alph = gf(2,m)`, and then you make the call `log(alph)`, it will return 1 because you gave it α^1 (MATLAB was smart enough to know that you gave it a Galois field element, so it used the appropriate definition of the logarithm). To see the full list of "overridden" functions, type `help gfhelp`. Of course you already expect that + (addition), $\star$ (multiplication), and ^ (exponentiation), have all been "overridden."
- As indicated on p. 247 of Moon's book, to compute the syndromes, you simply evaluate $S_j = r(\alpha^j)$ for $j = 1, 2, \cdots, 2t$. Based on the definitions above, you have everything you need, and everything is defined in $GF(2^m)$ with the supporting functions.
- If the syndromes are all zero then there are no errors, you're done.

- Wicker asks you to form the polynomial $S(x)$, which he defines for you. His algorithm uses this polynomial plus 1, i.e., $1 + S(x)$.

- Follow Wicker's algorithm, its pretty straightforward. In Step 2, you have to multiply two polynomials, do you remember how to multiply polynomials in MATLAB? Hopefully the function you need is on the list of "overridden" functions! In Step 3, adding the two polynomials looks pretty simple, but its harder than it needs to be because they are MATLAB vectors with (possibly) different lengths, so you'll need to take a few extra steps there.

- In my implementation, after the loop was done executing, the MATLAB vector holding the final polynomial $\Lambda(x)$ sometimes had extra high-order terms that were *zero*. Thus, it made it seem like the polynomial had a higher degree than was truly the case. I just looked for zero-valued terms at the end of the vector and deleted them. This solved the problem.

- At the end of the algorithm, you have to find the roots of $\Lambda(x)$ (Gee, if only there was a function called `roots` that was "overridden" to work with Galois fields). If there aren't any roots, or if there are more than t roots, then you have an uncorrectable error pattern (i.e., decoder failure). Otherwise, the logarithm of the roots (an integer in the range $0, 1, \cdots, 2^m - 2$) indicates the location of the errors in the received vector. Correct the errors and you are done.

Wicker pp 211-214

9.1.2 Berlekamp's Algorithm for Binary BCH Codes

Berlekamp's algorithm is much more difficult to understand than Peterson's approach, but results in a substantially more efficient implementation. The complexity of Peterson's technique increases with the square of the number of errors corrected, providing an efficient implementation for binary BCH decoders that correct up to 6 or 7 errors. The complexity of Berlekamp's algorithm increases linearly, allowing for construction of efficient decoders that correct dozens of errors. In this section Berlekamp's decoding algorithm for binary BCH codes is presented without proof. The reader who is interested in the full treatment is referred to Berlekamp [Ber1].

We begin by defining an infinite-degree syndrome polynomial

$$S(x) = S_1 x + S_2 x^2 + \cdots + S_{2t} x^{2t} + S_{2t+1} x^{2t+1} + \cdots \tag{9-14}$$

Clearly we do not know all of the coefficients of $S(x)$, but fortunately the first $2t$ coefficients are entirely sufficient. $S(x)$ is made into an infinite-degree polynomial so that it can be treated as a **generating function**. Define a third polynomial as follows.

$$\begin{aligned}\Omega(x) &\triangleq [1 + S(x)]\Lambda(x) \\ &= (1 + S_1 x + S_2 x^2 + \cdots)(1 + \Lambda_1 x + \Lambda_2 x^2 + \cdots) \\ &= 1 + (S_1 + \Lambda_1)x + (S_2 + \Lambda_1 S_1 + \Lambda_2)x^2 + (S_3 + \Lambda_1 S_2 + \Lambda_2 S_1 + \Lambda_3)x^3 + \cdots \\ &= 1 + \Omega_1 x + \Omega_2 x^2 + \cdots\end{aligned} \tag{9-15}$$

$\Omega(x)$ is called the **error magnitude polynomial** and is useful in nonbinary decoding. For now we will simply note that if the syndrome and error locator polynomials are to satisfy Eq. (9-7), then the odd-indexed coefficients of $\Omega(x)$ must be zero. Given that we know only the first $2t$ coefficients of $S(x)$, the decoding problem then becomes one of finding a polynomial $\Lambda(x)$ of degree less than or equal to t that satisfies

$$[1 + S(x)]\Lambda(x) \equiv (1 + \Omega_2 x^2 + \Omega_4 x^4 + \cdots + \Omega_{2t} x^{2t}) \bmod x^{2t+1} \tag{9-16}$$

Berlekamp's algorithm proceeds iteratively by breaking Eq. (9-16) down into a series of smaller problems of the form

$$[1 + S(x)]\Lambda^{(2k)}(x) \equiv (1 + \Omega_2 x^2 + \Omega_4 x^4 + \cdots + \Omega_{2k} x^{2k}) \bmod x^{2k+1} \tag{9-17}$$

where k runs from 1 to t. A solution $\Lambda^{(0)}(x) = 1$ is first assumed and tested to see if it works for the case $k = 1$. If it does work, we proceed to $k = 2$; otherwise, a correction factor is computed and added to $\Lambda^{(0)}$, creating a new solution $\Lambda^{(2)}(x)$. The genius of the algorithm lies in the computation of the correction factor. It is designed so that the new solution will work not only for the current case, but for all previous values of k as well. Once the algorithm concludes, the polynomial $\Lambda^{(2t)}(x)$ is a solution for all t of the expressions in Eq. (9-7).

Moon Eq (6.9)

Berlekamp's algorithm for decoding binary BCH codes [Ber1]

1. Set the initial conditions: $k = 0, \Lambda^{(0)}(x) = 1, T^{(0)} = 1$.
2. Let $\Delta^{(2k)}$ be the coefficient of x^{2k+1} in the product $\Lambda^{(2k)}(x)[1 + S(x)]$.
3. Compute

$$\Lambda^{(2k+2)}(x) = \Lambda^{(2k)}(x) + \Delta^{(2k)}[x \cdot T^{(2k)}(x)]$$

4. Compute

$$T^{(2k+2)}(x) = \begin{cases} x^2 T^{(2k)}(x) & \text{if } \Delta^{(2k)} = 0 \text{ or if } \deg[\Lambda^{(2k)}(x)] > k \\ \dfrac{x\Lambda^{(2k)}(x)}{\Delta^{(2k)}} & \text{if } \Delta^{(2k)} \neq 0 \text{ and } \deg[\Lambda^{(2k)}(x)] \leq k \end{cases}$$

5. Set $k = k + 1$. If $k < t$, then go to 2.
6. Determine the roots of $\Lambda(x) = \Lambda^{(2t)}(x)$. If the roots are distinct and lie in the right field, then correct the corresponding locations in the received word and STOP.
7. Declare a decoding failure and STOP.

Example 9-3—Double-Error Correction Using Berlekamp's Algorithm

In Example 9-1 we considered a narrow-sense double-error-correcting code of length 31 with generator polynomial = $1 + x^3 + x^5 + x^6 + x^8 + x^9 + x^{10}$. Let's repeat the example, but this time use Berlekamp's algorithm. The binary vector and associated polynomial

$$\mathbf{r} = (0010000110011000000000000000000)$$

$$\updownarrow$$

$$r(x) = x^2 + x^7 + x^8 + x^{11} + x^{12}$$

provide the following syndrome polynomial.

$$S(x) = \alpha^7 x + \alpha^{14} x^2 + \alpha^8 x^3 + \alpha^{28} x^4$$

Applying Berlekamp's algorithm, we obtain the following sequence of solutions to Eq. (9-17).

k	$\Lambda^{(2k)}(x)$	$T^{(2k)}(x)$	$\Delta^{(2k)}$
0	1	1	α^7
1	$1 + \alpha^7 x$	$\alpha^{24} x$	α^{22}
2	$1 + \alpha^7 x + \alpha^{15} x^2$	—	—

$\Lambda^{(4)}(x) = 1 + \alpha^7 x + \alpha^{15} x^2$ is, of course, the same error locator polynomial obtained in Example 9-1. ■

Example 9-4—Triple-Error Correction Using Berlekamp's Algorithm

As in Example 9-2, we let the code **C** be the triple-error-correcting narrow-sense binary BCH code with length 31. The generator polynomial

$$g(x) = 1 + x + x^2 + x^3 + x^5 + x^7 + x^8 + x^9 + x^{10} + x^{11} + x^{15}$$

has six consecutive roots $\{\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6\}$, where α is primitive in GF(32). Let the received polynomial be

$$r(x) = 1 + x^9 + x^{11} + x^{14}$$

With a bit of effort the syndrome polynomial is seen to be

$$S(x) = x + x^2 + \alpha^{29} x^3 + x^4 + \alpha^{23} x^5 + \alpha^{27} x^6$$

Berlekamp's algorithm then proceeds as follows.

k	$\Lambda^{(2k)}(x)$	$T^{(2k)}(x)$	$\Delta^{(2k)}$
0	1	1	1
1	$1 + x$	x	α^3
2	$1 + x + \alpha^3 x^2$	$\alpha^{28} x + \alpha^{28} x^2$	α^{20}
3	$1 + x + \alpha^{16} x^2 + \alpha^{17} x^3$	—	—

The error locator polynomial is then

$$\begin{aligned}\Lambda(x) &= 1 + x + \alpha^{16} x^2 + \alpha^{17} x^3 \\ &= (1 + \alpha^{13} x)(1 + \alpha^{16} x)(1 + \alpha^{19} x)\end{aligned}$$

indicating errors at the positions corresponding to α^{13}, α^{16}, and α^{19}. The corrected received word is then

$$r(x) = 1 + x^9 + x^{11} + x^{13} + x^{14} + x^{16} + x^{19} = (x^4 + x + 1)g(x)$$ ■