Linear Block Coding using MATLAB

Example: Generator Matrix for (7,4) Hamming Code

- Note: We have permuted the order of the information and parity check bits from table in to correspond with the convention used in Proakis for the Generator Matrix - this is still the same code.

$$\mathbf{G} = \left[\begin{array}{ccc|cccc} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right]$$

- Example $\underline{c} = [1001011] = [1011] \cdot \mathbf{G}$

Chk this with matlab

If $G=[I_k|P]$: then c=[1011100]

```
>> P=[1 1 0;0 1 1;1 1 1;1 0 1]

P =

     1     1     0
     0     1     1
     1     1     1
     1     0     1

>> G=[P eye(4)]

G =

     1     1     0     1     0     0     0
     0     1     1     0     1     0     0
     1     1     1     0     0     1     0
     1     0     1     0     0     0     1

>> c=x*G

c =

     3     2     2     1     0     1     1
```

[1 0 0 1 0 1 1]

Parity Check Matrices

- For each Generator Matrix **G**, it is possible to find a corresponding parity check matrix **H**
- For a systematic representation:

$$\mathbf{H} = \left[\mathbf{I}_{n-k} \middle| \overline{\mathbf{P}'}\right]$$

$$= \left[\begin{array}{cccc|ccc} 1 & 0 & \cdots & 0 & \overline{p}_{0,0} & \cdots & \overline{p}_{k-1,0} \\ 0 & 1 & & \vdots & \overline{p}_{0,1} & \cdots & \overline{p}_{k-1,1} \\ \vdots & & \ddots & 0 & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & \overline{p}_{0,n-k-1} & \cdots & \overline{p}_{k-1,n-k-1} \end{array}\right]$$

Parity Check Matrix for (7,4) Hamming Code

$$\mathbf{H} = \left[\begin{array}{ccc|cccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right]$$

```
>> H=[eye(3) P']

H =

     1     0     0     1     0     1     1
     0     1     0     1     1     1     0
     0     0     1     0     1     1     1
```

Parameters for Linear Block Codes

This subsection describes the items that you might need in order to process [n,k] cyclic, Hamming, and generic linear block codes. The table below lists the items and the coding techniques for which they are most relevant.

Parameters Used in Block Coding Techniques

Parameter	Block Coding Technique
Generator Matrix	Generic linear block
Parity-Check Matrix	Generic linear block
Generator Polynomial	Cyclic
Decoding Table	Generic linear block, Hamming

Generator Matrix

The process of encoding a message into an [n,k] linear block code is determined by a k-by-n generator matrix G. Specifically, the 1-by-k message vector v is encoded into the 1-by-n codeword vector vG. If G has the form $[I_k\ P]$ or $[P\ I_k]$, where P is some k-by-(n-k) matrix and I_k is the k-by-k identity matrix, then G is said to be in *standard form*. (Some authors, e.g., Clark and Cain [2], use the first standard form, while others, e.g., Lin and Costello [3], use the second.) Most functions in this toolbox assume that a generator matrix is in standard form when you use it as an input argument.

Some examples of generator matrices are in the next section, Parity-Check Matrix

Parity-Check Matrix

Decoding an [n,k] linear block code requires an (n-k)-by-n parity-check matrix H. It satisfies $GH^{tr} = 0 \pmod 2$, where H^{tr} denotes the matrix transpose of H, G is the code's generator matrix, and this zero matrix is k-by-(n-k). If $G = [I_k \; P]$ then $H = [-P^{tr} \; I_{n-k}]$. Most functions in this toolbox assume that a parity-check matrix is in standard form when you use it as an input argument.

The table below summarizes the standard forms of the generator and parity-check matrices for an [n,k] binary linear block code.

Type of Matrix	Standard Form	Dimensions
Generator	$[I_k \; P]$ or $[P \; I_k]$	k-by-n
Parity-check	$[-P' \; I_{n-k}]$ or $[I_{n-k} \; -P']$	(n-k)-by-n

I_k is the identity matrix of size k and the ' symbol indicates matrix transpose. (For *binary* codes, the minus signs in the parity-check form listed above are irrelevant; that is, -1 = 1 in the binary field.)

Examples. In the command below, `parmat` is a parity-check matrix and `genmat` is a generator matrix for a Hamming code in which [n,k] = [2^3-1, n-3] =[7,4]. Notice that `genmat` has the standard form [P I_k].

```
[parmat,genmat] = hammgen(3)
parmat =

     1     0     0     1     0     1     1
     0     1     0     1     1     1     0
     0     0     1     0     1     1     1


genmat =

     1     1     0     1     0     0     0
     0     1     1     0     1     0     0
     1     1     1     0     0     1     0
     1     0     1     0     0     0     1
```

The next example finds parity-check and generator matrices for a [7,3] cyclic code. The `cyclpoly` function is mentioned below in Generator Polynomial.

```
genpoly = cyclpoly(7,3);
[parmat,genmat] = cyclgen(7,genpoly)
parmat =

     1     0     0     0     1     1     0
     0     1     0     0     0     1     1
     0     0     1     0     1     1     1
     0     0     0     1     1     0     1

genmat =

     1     0     1     1     1     0     0
     1     1     1     0     0     1     0
     0     1     1     1     0     0     1
```

Decoding Table

A decoding table tells a decoder how to correct errors that might have corrupted the code during transmission. Hamming codes can correct any single-symbol error in any codeword. Other codes can correct, or partially correct, errors that corrupt more than one symbol in a given codeword.

This toolbox represents a decoding table as a matrix with `n` columns and `2^(n-k)` rows. Each row gives a correction vector for one received codeword vector. A Hamming decoding table has `n+1` rows. The `syndtable` function generates a decoding table for a given parity-check matrix.

Example: Using a Decoding Table. The script below shows how to use a Hamming decoding table to correct an error in a received message. The `hammgen` function produces the parity-check matrix, while the `syndtable` function produces the decoding table. The transpose of the parity-check matrix is multiplied on the left by the received codeword, yielding the *syndrome*. The decoding table helps determine the correction vector. The corrected codeword is the sum (modulo 2) of the correction vector and the received codeword.

```
% Use a [7,4] Hamming code.
m = 3; n = 2^m-1; k = n-m;
parmat = hammgen(m); % Produce parity-check matrix.
trt = syndtable(parmat); % Produce decoding table.
recd = [1 0 0 1 1 1 1] % Suppose this is the received vector.
syndrome = rem(recd * parmat',2);
syndrome_de = bi2de(syndrome,'left-msb'); % Convert to decimal.
disp(['Syndrome = ',num2str(syndrome_de),...
      ' (decimal), ',num2str(syndrome),' (binary)'])
corrvect = trt(1+syndrome_de,:) % Correction vector
% Now compute the corrected codeword.
correctedcode = rem(corrvect+recd,2)
```

The output is below.

```
recd =

     1     0     0     1     1     1     1
```

```
Syndrome = 3 (decimal), 0  1  1 (binary)
```

```
corrvect =

     0     0     0     0     1     0     0
```

```
correctedcode =

     1     0     0     1     0     1     1
```

```
% Use a [7,4] Hamming code.
m = 3; n = 2^m-1; k = n-m;
parmat = hammgen(m); % Produce parity-check matrix.
trt = syndtable(parmat); % Produce decoding table.
recd = [1 0 0 1 1 1 1] % Suppose this is the received vector.
syndrome = rem(recd * parmat',2);
syndrome_de = bi2de(syndrome,'left-msb'); % Convert to decimal.
disp(['Syndrome = ',num2str(syndrome_de),...
      ' (decimal), ',num2str(syndrome),' (binary)'])
corrvect = trt(1+syndrome_de,:) % Correction vector
% Now compute the corrected codeword.
correctedcode = rem(corrvect+recd,2)
```