Digital Modulations

Q: Recall that a digital comm. system conveys digital info (bits) across a channel. How is this done?

A: We've actually answered this question partially. We have to convert the digital info (a sequence of bits) into some signals (analog) that we can send through the channel. This process is called digital modulation. Of course, at the receiver, we have to convert the received signal back to a sequence of bits by the reverse process known as digital demodulation.
The design of digital modulation/demodulation (MODEM) schemes depends on many factors. The two major factors are the:

(i) communication channel &
(ii) bit rate.

**Line Coding**

First let us consider a baseband (LP) channel whose BW is much bigger than that of the transmitted signal (more precise discussion later), i.e. the bit rate is lower than the BW of channel. Typical applications that correspond to this situation include low-rate interface to computers, ethernet with coaxial cables, etc.

For this channel, the typical modulation technique is line coding. Line coding refers to the modulation technique that uses a pulse to represent the bit value "1" and another (different) pulse to represent the bit value "0". The pulse rate is that same as the bit rate.
Common **line codes**:

(a) Punched Tape

(b) Unipolar NRZ

(c) Polar NRZ

(d) Unipolar RZ

(e) Bipolar RZ

(f) Manchester NRZ

\[ N = \text{Non}, \quad R = \text{Return}, \quad Z = \text{Zero} \]
PSDs of line codes

We are interested in the PSDs of line codes to determine properties such as BW and DC component etc.

Many line code signals can be written as

\[ y(t) = p(t) * \sum_{n=-\infty}^{\infty} a_n \delta(t-nT_b) \]

where

- \( p(t) \) is the pulse shape used in line coding.
- \( T_b \) is the bit duration (pulse duration).
- \( Y(b) = \text{bit rate} = \text{pulse rate} \)

\( a_n \)'s are the values that represent the bits.

(e.g. for NRZ: \( a_n \in \{1,0\} \) and \( p(t)=\text{Rect}(\frac{t}{T_b}) \))

Thus

\[ X(t) \xrightarrow{\text{filter}} p(t) \xrightarrow{\text{convolution}} y(t) \]

and

\[ S_y(w) = |P(w)|^2 S_x(w) \]

It can be shown (see text section 7.2 for details) that

\[ R_X(t) = \frac{1}{T_b} \sum_{n=-\infty}^{\infty} R_n \delta(t-nT_b) \]

\[ S_x(w) = \frac{1}{T_b} \sum_{n=-\infty}^{\infty} R_n e^{-jnwT_b} \]

\[ = \frac{1}{T_b} \left( R_0 + 2 \sum_{n=1}^{\infty} R_n \cos(nwT_b) \right) \quad (R_n=R_{-n}) \]

where

\[ R_n = \lim_{N \to \infty} \frac{1}{N} \sum_{k=-N}^{N} A_k e^{j2\pi k nth} \]
Example: Assuming it is equally likely for a bit to be "0" or "1".

(i) Polar NRZ:

\[ a_n = \{ \pm 1 \} \quad , \quad p(t) = A \text{rect} \left( \frac{t}{T_b} \right) \]

\[ R_0 = \lim_{N \to \infty} \frac{1}{N} \sum_{n=0}^{N} a_n^2 = \lim_{N \to \infty} \frac{1}{N} \left( \frac{N}{2} \right) = \frac{1}{2} \]

\[ R_n = \lim_{N \to \infty} \frac{1}{N} \sum_{n=0}^{N} a_n a_{n+m} = \lim_{N \to \infty} \frac{1}{N} \left( \frac{N}{2} + \frac{N}{2} \right) = 0 \quad , \quad n \neq 1 \]

Then

\[ S_x(w) = \frac{1}{T_b} \quad \text{and} \quad P(w) = A T_b \sin \left( \frac{w T_b}{2} \right) \]

\[ S_y(w) = S_x(w) |P(w)|^2 = \frac{2}{T_b} \sin^2 \left( \frac{w T_b}{2} \right) \]

(ii) Unipolar NRZ:

\[ a_n = \{ 0, 1 \} \quad , \quad p(t) = A \text{rect} \left( \frac{t}{T_b} \right) \]

\[ R_0 = \lim_{N \to \infty} \frac{1}{N} \left[ \frac{N}{2}(1) + \frac{N}{2}(0) \right] = \frac{1}{2} \]

\[ R_n = \lim_{N \to \infty} \frac{1}{N} \left[ \frac{N}{4}(1) + \frac{N}{4}(0) \right] = \frac{1}{4} \quad , \quad n \neq 1 \]

\[ S_x(w) = \frac{1}{4 T_b} + \frac{1}{4 T_b} \sum_{n=-\infty}^{\infty} e^{-j nw T_b} \]

\[ = \frac{1}{4 T_b} + \frac{2 \pi}{4 T_b^2} \sum_{n=-\infty}^{\infty} \delta \left( w - \frac{2 \pi n}{T_b} \right) \]

Thus

\[ S_y(w) = S_x(w) |P(w)|^2 = \frac{A^2 T_b}{4} \sin^2 \left( \frac{w T_b}{2} \right) \left[1 + \frac{2 \pi}{T_b} \sum_{n=-\infty}^{\infty} \delta \left( w - \frac{2 \pi n}{T_b} \right) \right] \]

(iii) Polar RZ:

\[ a_n = \{ \pm 1 \} \quad , \quad p(t) = A \text{rect} \left( \frac{2T}{T_b} \right) \]

\[ S_y(w) = \frac{A^2 T_b}{4} \sin^2 \left( \frac{w T_b}{4} \right) \]

(iv) Unipolar RZ:

\[ a_n = \{ 0, 1 \} \quad , \quad p(t) = A \text{rect} \left( \frac{2T}{T_b} \right) \]

\[ S_y(w) = \frac{A^2 T_b}{4} \sin^2 \left( \frac{w T_b}{4} \right) \left[1 + \frac{2 \pi}{T_b} \sum_{n=-\infty}^{\infty} \delta \left( w - \frac{2 \pi n}{T_b} \right) \right] \]
(v) Manchester. NRZ: \( a(t) = \{ \pm 1 \} \), 
\( p(t) = \left[ \text{rect} \left( \frac{2t}{T_b} + \frac{1}{2} \right) - \text{rect} \left( \frac{2t}{T_b} - \frac{1}{2} \right) \right] \).

\[
|P(f)|_2 = \frac{1}{T_b} \left( 1 - \frac{\sin \left( \frac{\pi f T_b}{2} \right)}{\pi \frac{2T_b}{T_c}} \right)^2
\]

\[
S_g(w) = \frac{\pi^2}{2} \left( 1 - \cos \left( \frac{\pi f T_b}{2} \right) \right)^2
\]

Observations: (i) Both NRZ, Unipolar NRZ, Polar RZ, Unipolar RZ all have DC component. Manchester NRZ does not.

(ii) BWs of RZ's are higher than BWs of NRZ's.

(iii) Unipolar RZ has impulses in PSD while others do not.

**Pulse-Amplitude Modulation (PAM)**

In the line coding techniques described above, we send 1 bit per pulse. Thus, the bit rate (\# of bits per second) is the same as the pulse rate. If we want to send more bits per second through the channel, one method is to reduce the pulse duration \( T_b \), i.e., increase the bit rate. However, this will increase the bandwidth of the transmitted signal.
In this scheme, the digital takes 2 bits of information, say -3V, -1V, +1V, +3V to represent 2 voltage levels, we can employ the idea in the following way: if a single bit is used, we can employ a positive and a negative voltage level (say, +1V) to represent the bit value (1/0). Example:

Another way is to employ a multi-level signal, which will be filtered and cause ISI and crosstalk. This will distort the components of the signal. If the bandwidth of the channel becomes larger than the maximum frequency allowed by the channel, then the back亡idth of the signal becomes larger.
The 4 voltage levels depending on the bit values. We can use, say, the Gray mapping:

-3V -1V +1V +3V
  ↑ ↑ ↑ ↑
  01 00 10 11

The transmitter then sends out a pulse of the corresponding voltage. In this way, we transmit 2 bits per pulse with increasing the bandwidth of the transmitted signal. Here we double the bit rate.

At receiver, the reverse process is performed to map the received voltage back to one of the 4 possible bit patterns. Instead of using a single threshold, we have to employ 3 thresholds in this case: let V be the received voltage of the pulse,

- if $V \leq -2V$, output bit pattern "01"
- if $-2V < V < 0V$, output bit pattern "00"
- if $0V \leq V < 2V$, output bit pattern "10"
- if $V \geq 2V$, output bit pattern "11"
This method can be, of course, generalized to any
value of k to carry k info bits per pulse.

This modulation scheme is known as pulse amplitude
modulation (PAM).

The advantage of PAM is that we can carry
more data bits through the channel. The
disadvantage of PAM is that we have to
increase the transmission power if we want to
make the bit error prob. the same as in the
case of NRZ. In other words, if we don't
increase the transmission power (by scaling back
the voltage levels), we are more prone to error
with PAM than with NRZ.

Q: How about bandpass channels? (a preview here).

A: For bandpass channels, the idea is to freq-shift
the baseband digitally modulated signal to the passband
of the channel by multiplying it with a carrier.
The same scheme employed for double-sideband
suppressed carrier modulation directly applies here.
Intersymbol Interference & Equalization

Q: From previous discussions, we know that the impulse response of a comm. channel characterizes the channel. But what kind of impulse responses are common? How do these common impulse responses affect the transmitted signal?

A: There are as many different types of impulse responses as the number of different comm. channels. However most channels (their impulse responses) tend to smear the transmitted pulses. For example, if NRZ signal is used, the transmitted pulses will be rectangular. Most channels smear these rectangular pulses and the resulting received pulses will no longer be rectangular.

Q: Is the "smearing" of the transmitted pulses bad?

A: Yes, it can be pretty bad. The "smearing" of the transmitted pulses has two detrimental effects:

(i) The received pulses are not exactly in the same shape of the transmitted pulses. This may cause problem in the decision process.

(But this is not too big a problem.)
(ii) If the impulse response of the channel is longer than a bit duration, then the transmitted pulse will be smeared to such an extent that its effect will extend to the following bits, causing what we call **intersymbol interference (ISI)**. This is a bad but common phenomenon in practical channels.

E.g. The following figure shows the impulse response of a typical telephone line channel (Proakis, p.)

![Graph showing impulse response with Time (ms) on the x-axis and Amplitude on the y-axis.]
Q: How about an example to illustrate ISI pictorially?

A: Of course. To make our drawings simple, let's consider the following simple but somewhat unrealistic scenario:

Fig:
Suppose that we are sending 5 bits 10110 using polar NRZ with a bit duration of T see. The comm. channel considered is a “2-ray” channel, meaning that there are 2 transmission paths from the transmitter to the receiver. For example, one path can be a direct line-of-sight path from Tx to Rx. The other path can be a reflected path off a distant mountain to the Rx.

Assume that the impulse response of the channel

\[ h(t) = s(t) + 0.8 s(t-T) \]

Direct LOS path \hspace{1cm} Reflected path (delayed by T & lost same power)

\[ (t) \]

NRZ signal

Impulse response
First, let's temporarily neglect the thermal noise. We know that the received signal is simply the convolution of $h(t)$ and the transmitted NRZ signal.

We can, of course, perform the convolution directly based on the plots of the two signals on the previous page. However, due to the linearity of the convolution operation, we can perform the convolution for each pulse in the NRZ signal separately and then add up the result. It turns out that this approach is more illustrative for our purpose here (showing the effect of ISI).
We see that the channel "smears" the rectangular pulses and the effect of a bit is spread to the following bit. This spreading of the effect of one bit to the next can be good or bad. For example, the effect of the 1st bit is spread to the 2nd bit causing the voltage level of the pulse corresponding to the 2nd bit (value = 0) to reduce from -1V to -0.2V. On the other hand, the effect of the 3rd bit is spread to the 4th bit, causing the voltage level of the pulse corresponding to the 4th bit to go up from 1V to 1.8V.

At first, it seems that ISI is not a problem since we can still get correct decision by examining whether the voltage level is positive or negative. However, this is not exactly true since we have to add back the omni-presence thermal noise to the received signal. Recall that noise causes errors and the effect of noise is more significant when the voltage level of the NRZ signal is smaller. For example, consider the 2nd bit, without the ISI effect, we can tolerate a noise sample with a magnitude as large as 1V, assuming that we take the 1st sample in every bit to make the decision. However, with ISI, the reduction in voltage level limits the tolerance against noise to only 0.2V. Here we are more likely to make
A similar argument applies to the case of summing up all the samples of a bit to make the decision. As a result, with ISI, we are more likely to make worse erratic decisions (the bad effect of ISI actually dominates the good effect).

Q: Is there any easy way to visualize the effect of ISI at least qualitatively?

A: A simple way to do so is to generate an "eye pattern" of the Rx signal by displaying the Rx signal using an oscilloscope with the sweeping rate set to be half of the bit rate (i.e., the sweeping period is twice the bit period) and starting the sweep in the middle of a bit period. (This can be done by adjusting the horizontal position of the oscilloscope display).

E.g.) NRZ signal connected directly to osc.

```
\begin{center}
\begin{tikzpicture}
\begin{scope}
\draw (0,0) -- (4,0);
\draw (0,1) -- (2,1) -- (2,2) -- (4,2);
\draw (0,3) -- (2,3) -- (2,4) -- (4,4);
\end{scope}
\begin{scope}[xshift=5cm]
\draw (0,0) -- (4,0);
\draw (0,1) -- (2,1) -- (2,2) -- (4,2);
\draw (0,3) -- (2,3) -- (2,4) -- (4,4);
\end{scope}
\end{tikzpicture}
\end{center}
```

NRZ signal

sweeping voltage
Eq.1) Connect NRZ signal directly to osc.

\[ \text{eye opening} \]

\[ 1.8V \]

\[ 1V \]

\[ 0.2V \]

\[ 0.2V \] & eye opening

\[ 1V \]

\[ -1.8V \]

Eq.2) Connect the received signal in the "2-ray" channel example previously (without the noise).

Note that the actual received signal contains noise and hence the eye patterns will look noisy.

The eye opening tells us the noise margin, which is the amount of noise that the signalling scheme can tolerate over a specific channel. Noise margin is \( \frac{1}{2} \) of the width of the eye opening.

Comparing Eq.1 and Eq.2, we see that the "2-ray" channel causes ISI which reduces the noise margin from the original 1V to 0.2V.
Q: Can we generate the eye patterns of a signal using MATLAB?

A: Yes, of course.

Hint: use the commands "reshape" and "plot".

Q: OK, so ISI is bad. Is there a way to reverse the detrimental effect of ISI?

A: There are many ways to reduce the effect of ISI. These methods are usually referred to as equalization collectively. Here we describe one of the most intuitive equalization methods known as zero-forcing (ZF) equalization.

First, we pretend that noise is not present in the system and hence the received signal is simply the transmitted signal convolved with the impulse response of the channel, i.e.,

\[ r(t) = h(t) * s(t) \]
Eq.) To explain the concept of ZF equalization in a simple manner, let's once again go back to the "2-ray" channel example. In this case, \( r(t) = s(t) + 0.8 s(t-T) \). Hence

\[ r(t) = s(t) \ast r(t) = s(t) + 0.8 s(t-T). \]

From the equation, we can see that the effect of the previous bit spreads over to the current bit.

What we want to do is to remove this ISI to get a clean copy of \( s(t) \).

By rearranging the equation, we have

\[ s(t) = r(t) - 0.8 s(t-T). \]

Therefore, if we knew \( s(t-T) \), we could simply subtract it from the received signal to get the clean \( s(t) \).

However, we don't know \( s(t-T) \). \( s(t-T) \) is just a delayed version of \( s(t) \). If we knew \( s(t-T) \), we knew \( s(t) \) and there is no need for communications.

Nevertheless, there is a way out: Notice that

\[ s(t-T) = r(t-T) - 0.8 s(t-2T). \]

Putting this back into the equation above, we have

\[ s(t) = r(t) - 0.8 r(t-T) + 0.64 s(t-2T). \]

We can continue to apply the same trick to \( s(t-2T) \) and so on...
Eventually, we will get

\[ s(t) = r(t) - 0.8 r(t-T) + 0.8^2 r(t-2T) - 0.8^3 r(t-3T) + \cdots \]

\[ = \sum_{k=0}^{\infty} (-0.8)^k r(t-kT) \]

\[ = r(t) \ast g(t) \]

where \( g(t) = \sum_{k=0}^{\infty} (-0.8)^k s(t-kT) \).

Therefore, we can obtain a clean (assuming no noise) copy of \( s(t) \) by weighting and combining delayed versions of the received signal only. However, we need infinitely many delayed versions of \( r(t) \). Fortunately, we can see from the equation above that the effect of delayed versions of \( r(t) \) with large delays is small due to the exponential decaying factor \( (0.8)^k \). This translates to the physical intuition that the current bit only affects the next bit. Hence, we only need a finite number of delayed versions of \( r(t) \) to approximately remove the ISI in practice.

Q: Can we generalize this idea to an arbitrary impulse response?

A: Yes. To do so, we notice that in the above example

\[ g(t) \ast r(t) = s(t) \]
This is exactly what we need to generalize the result. Consider the following figure.

\[ s(t) \rightarrow h(t) \rightarrow R(t) \rightarrow g(t) \rightarrow y(t) \]  
no ISI.

The transmitted signal \( s(t) \) goes through the channel and results in ISI. If we filter the received signal (with ISI by using a filter with impulse response \( g(t) \) such that \( g(t) \ast R(t) = s(t) \)), then the output of the filter will be a clean copy of \( s(t) \) without ISI, i.e., we completely remove ISI from the received signal, or we force the filter output to give zero ISI. This is why this technique is called ZF equalization.

To prove the above statement, we need

\[ y(t) = g(t) \ast r(t) \]
\[ = g(t) \ast R(t) \ast s(t) \]
\[ = s(t) \ast s(t) \]
\[ = s(t) \]

Q.E.D.

Therefore, all we need to do is to find the filter \( g(t) \) which is called the inverse filter of \( R(t) \). The
Q: ZF equalization seems to be perfect, isn't it so?

A: Well, the answer is no. ZF equalization has 2 drawbacks:

1) In general, the process of deconvolution is not simple and in some cases, the inverse filter $g(t)$ of the channel impulse response $R(t)$ may be unrealizable (unstable).

2) We have completely ignored the presence of noise. It turns out that in some cases, although the inverse filter $g(t)$ can completely remove the ISI, it may amplify the thermal noise. It is also possible that the effect of amplifying the thermal noise can be even worse than that of the ISI. As a result, we may not be better off by removing the ISI using $g(t)$ at all. In this case, we need some other equalization strategy to strike a balance between removing the ISI and amplifying the thermal noise.
Error Control Coding

- When errors occur during the demodulation process, the usual solution is for the receiver to ask the transmitter to send the signal again (assuming that there is a feedback channel from the receiver to the transmitter).

- This approach is generally referred to as automatic repeat request (ARQ).

- A more careful thought reveals that the receiver needs to possess the ability to detect the presence of errors in order to be able to perform ARQ.

- Special signal (coding) designs are needed for the transmitted signal. The resulting coding technique is called error detection coding.

- Example: Error-detecting repetition code.

  Let us consider the simplest error detecting code. Suppose instead of sending an information bit directly, (with suitable modulation techniques), we copy the bit and send the copy and the original bit together.
That is, \[ 0 \rightarrow 00, \quad 1 \rightarrow 11. \]

- If the received bit pair is either 00 or 11, then we decide that there is no error and can decode the information bit value correspondingly (0 or 1 respectively).

- If the received bit pair is either 01 or 10, then we know that one of the bits is in error (and hence can ask the transmitter for retransmission).

- If both of the transmitted bits are in error, then the receiver will not be able to detect this erroneous event and decode incorrectly.

In summary, the simple error-detecting repetition can detect a single error. The price to pay is a 50% reduction in transmission efficiency (half of the original info rate).

- Example: Single (even) parity-check code
  The coding efficiency of the previous repetition code is pretty low (detects a single error by halving the bit rate).
  In particular, if the chance of having a bit error is not high, then a more common (efficient) error-detecting code is the single parity-check code.
To describe the parity check code, we need to first introduce the mechanism of modulo-2 arithmetic:

### Finite field of 2 elements \{0, 1\}

#### Addition

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

(XOR)

#### Multiplication

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

(AND)

(additive inverse)

Subtraction same as addition and negative is the element itself (Why?). Division is trivial since we only allow \(1/1 = 1\) and \(0/1 = 0\).

- Can also be thought of as "regular" arithmetic modulo 2.
  - For examples, \(1 + 1 = 2 = 0 \mod 2\), \(1 \times 1 = 1 \mod 2\), \(0 - 1 = -1 = 1 \mod 2\).

Going back to the single-parity check code:

We group \(k\) bits together into a block and then find the sum of the \(k\) info bits (modulo-2 sum).

The sum is called a parity bit:

\[
y = \sum_{i=1}^{k} x_i
\]

(part)

And we send \([x_1, x_2, x_3, \ldots, x_k, y]\).

Notice that \(y + \sum_{i=1}^{k} x_i = y + y = 0\).

This equation gives us a way to check for errors in the received bits.
At the receiver, we sum (mod 2) up all the
received bits.

(i) If the sum is 0, then we decide that there is no error
and discard the parity bit.

(ii) If the sum is 1, then we decide that there are errors
and ask for retransmission.

If there are in fact an odd number of errors, the above
procedures will detect the errors.
If there are in fact an even number of errors, the above
procedure will not be able to detect the errors.

In summary, the parity-check code is able to detect
guaranteed to detect a single error.
Since we add a single parity bit to every k info bit, the
transmission efficiency is high when k is large.
The choice of k depends on how likely an error occurs.
If the error prob is small, we can choose a large k.
If the error prob is not small, we choose a small k.

Often used in practice when the error prob. is small.
- When the feedback channel is not available or the cost of using it is high, the approach of forward error correction appears more appealing.

- In the FEC approach, we need a code that can correct transmission errors. Such code is called an error-correcting code.

  Example: Error-correcting repetition code.

  By this code, we send the info bit thrice, i.e.
  
  \[
  0 \rightarrow 000 \\
  1 \rightarrow 111
  \]

  The receiver decoding by majority vote; decides 0 (or 1) if the majority of the three received bits are 0 (or 1).

  One can easily see that this repetition code can correct a single transmission error.

  Incidentally, this error-correcting code can also be used for error detection. Notice that it can detect at most 2 errors. (But it cannot detect & correct errors simultaneously.)

  Generally, all error-correcting codes can be used for error detection.

  The coding efficiency of the repetition code is not high.
To improve coding efficiency, we can use the idea of parity check code before: code a block of K info bits together, rather than one bit at a time.

But first, let's revisit the error-correcting repetition code before from the viewpoint of vector space operations (modulo 2):

We can consider the vector \([111]\) as the basic vector for this code and the two possible codewords \([000]\) and \([111]\) are generated respectively by multiplying the basic vector \([111]\) with the information bit values 0 and 1.

In matrix notation, we have

\[
\begin{bmatrix} 0 \\ 1 \end{bmatrix} \cdot [111] = [000] \\
\text{info bit} \quad \text{basic vector} \quad \text{codeword}
\]

\[
[1] \cdot [111] = [111]
\]

Now consider decoding; the majority decoding algorithm written before can also be done by matrix operations:

Consider that there is at most a single transmission error and consider the matrix 

\[
\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}
\]
Correct Rx codeword: 
\[
\begin{bmatrix}
0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
1 & 0 \\
0 & 1 \\
1 & 1
\end{bmatrix}
= 
\begin{bmatrix}
0 & 0
\end{bmatrix}
\]
\[
\begin{bmatrix}
1 & 1 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 \\
0 & 1 \\
1 & 1
\end{bmatrix}
= 
\begin{bmatrix}
0 & 0
\end{bmatrix}
\]

1st Rx bit in error: 
\[
\begin{bmatrix}
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 \\
0 & 1 \\
1 & 1
\end{bmatrix}
= 
\begin{bmatrix}
1 & 0
\end{bmatrix}
\]
\[
\begin{bmatrix}
1 & 1 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 \\
0 & 1 \\
1 & 1
\end{bmatrix}
= 
\begin{bmatrix}
1 & 0
\end{bmatrix}
\]

2nd Rx bit in error: 
\[
\begin{bmatrix}
0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
1 & 0 \\
0 & 1 \\
1 & 1
\end{bmatrix}
= 
\begin{bmatrix}
0 & 1
\end{bmatrix}
\]
\[
\begin{bmatrix}
1 & 1 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 \\
0 & 1 \\
1 & 1
\end{bmatrix}
= 
\begin{bmatrix}
0 & 1
\end{bmatrix}
\]

3rd Rx bit in error: 
\[
\begin{bmatrix}
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 \\
0 & 1 \\
1 & 1
\end{bmatrix}
= 
\begin{bmatrix}
1 & 1
\end{bmatrix}
\]
\[
\begin{bmatrix}
1 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
1 & 0 \\
0 & 1 \\
1 & 1
\end{bmatrix}
= 
\begin{bmatrix}
1 & 1
\end{bmatrix}
\]

This analysis suggests the following syndrome decoding strategy:

1) Left-multiply the received codeword vector with the matrix 
\[
\begin{bmatrix}
1 & 0 \\
0 & 1
\end{bmatrix}
\]. The product vector is called the syndrome vector.

2) The syndrome vector tells us the location of the possible
single transmission error:
\[
\begin{bmatrix}
0 & 0
\end{bmatrix} \rightarrow \text{no error}
\]
\[
\begin{bmatrix}
1 & 0
\end{bmatrix} \rightarrow \text{1st bit (leftmost) in error}
\]
\[
\begin{bmatrix}
0 & 1
\end{bmatrix} \rightarrow \text{2nd bit (from left) in error}
\]
\[
\begin{bmatrix}
1 & 1
\end{bmatrix} \rightarrow \text{3rd bit (rightmost) in error}
\]

3) Reconstruct the transmit codeword (force decode the info bit)
by adding the error pattern to the received codeword.
For example, if the syndrome is \[\begin{bmatrix} 0 & 1 \end{bmatrix}\], the error pattern is \[\begin{bmatrix} 0 & 1 \end{bmatrix}\] (2nd bit in error). Then we can obtain the transmit codeword as:
\[\begin{bmatrix} 0 & 1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}\]
or \[\begin{bmatrix} 1 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}\]
- When there are two transmission errors, the syndrome vector will still be non-zero (not \(000\)). Thus we can use the above syndrome decoding procedure to detect two transmission errors. However, we won't be able to determine whether there are 2 errors or 1 error and the location(s) of the error(s).

- When all the transmitted bits are in error, the syndrome vector will be zero, and hence the errors can neither be detected nor corrected.

- Once again, the argument suggests the repetition code can be used as a single-error correcting code or a double-error detecting code. For channels that are more likely to generate a single error than 2 or 3 errors, one should use the repetition code for single-error correction. Otherwise, one should use the code for double-error detection.

Linear Block Codes.

- The vector space coding and syndrome decoding interpretation of the repetition code reveals a more general way (and hence could provide more efficient codes) of constructing error-control codes.
Consider the following identification (more like rotation) of the components of the discussion before:

\[ \begin{align*}
0 \text{ or } 1 & \rightarrow \mathbf{u} \quad \text{(information vector)} \\
1 \text{ or } 0 & \rightarrow \mathbf{g} \quad \text{(generator matrix)} \\
0000 \text{ or } 1111 & \rightarrow \mathbf{v} \quad \text{(codeword)} \\
1011 & \rightarrow \mathbf{h} \quad \text{(parity-check matrix)} \\
000, [01], [10], 11 & \rightarrow \mathbf{s} \quad \text{(syndrome vector)} \\
0000, 0001, \ldots & \rightarrow \mathbf{e} \quad \text{(error vector)}
\end{align*} \]

The encoding procedure is then \( \mathbf{v} = \mathbf{u} \mathbf{g} \).

The channel generates the received vector \( \mathbf{r} = \mathbf{v} + \mathbf{e} \).

The decoding procedure is then:

1. Syndrome generation: \( \mathbf{s} = \mathbf{r} \mathbf{h}^{\top} \)
2. Error pattern matching: \( \mathbf{s} \rightarrow \mathbf{\hat{e}} \) (table lookup).
3. Error correction: \( \hat{\mathbf{v}} = \mathbf{r} + \mathbf{\hat{e}} \)

In general, if we encode a block of \( k \) inf bits together to generate codewords of length \( n \) bits (\( n > k \) here; adding redundancy), the dimensions of the various coding components becomes:

- \( \mathbf{u} = 1 \times k \)
- \( \mathbf{v} = 1 \times n \)
- \( \mathbf{g} = k \times n \)
- \( \mathbf{h} = (n-k) \times n \)
- \( \mathbf{s} = 1 \times (n-K) \)
- \( \mathbf{E} = 1 \times n \)

This generalization gives us a general linear block code.
So the design of block codes boils down to the selection of $k, n, \text{ and } \delta$ ($H$ is determined by $\delta$). The obvious goal is to find a $\delta$ such that the ratio $\frac{k}{n}$ is as close to 1 as possible while being able to correct as many errors as possible.

This design is generally difficult and there are many constraints that exist. For example, if we want to correct 1 error, then we need at least $(n-k)$ different syndrome patterns. Since the syndrome vector is of length $n-k$, it can have at most $2^{n-k}$ different patterns. So $2^{n-k} \geq 1 + n$. That is, the ratio $\frac{k}{n} \leq 1 - \frac{\log_2(1+n)}{n}$, meaning that it cannot be too close to 1 unless $n$ is large.

But when $n$ is large, it will be more likely to have more than 1 single error and hence the single-error correcting capability may not be sufficient.

Nevertheless, many "good" ("efficient") block codes have been found and we will look at one famous (simple) example called the Hamming code later.
But first let us make one more observation from the repetition code example:

\[ P \]

identity \((1 \times 1)\)

Notice that the generator matrix \( G = \begin{bmatrix} 1 & 1 \end{bmatrix} \)

and the parity-check matrix \( H = \begin{bmatrix} 0 & 1 \end{bmatrix} \)

have some special forms.

Generalize this special forms, we have

\[ G = \begin{bmatrix} P & I_k \end{bmatrix} \]

\((R \times (n-k)) \times R \times R\) identity matrix

and

\[ H = \begin{bmatrix} I_k & P^T \end{bmatrix} \]

\((n-k) \times R \times (n-k)\) transpose of \( P \)

identity matrix

Linear block codes that have the generator and parity-check matrices in the forms above are called systematic codes.

The reason is that the codeword \( \mathbf{v} = \begin{bmatrix} \mathbf{p} & \mathbf{u} \end{bmatrix} \)

so one can decode by simply decoding the parity bits in front of the codeword to get back the info. bits.

Also, for systematic linear block codes, the above formula provides a simple way to obtain \( H \) from \( G \).

Notice that

\[ GH^T = \begin{bmatrix} P & I_k \end{bmatrix} \begin{bmatrix} I_k & P^T \end{bmatrix} = \begin{bmatrix} P + P \end{bmatrix} \]

identity matrix

This relationship can be used to determine \( H \) from \( G \) for linear block codes that are not systematic.
Example: $(7,4)$ Hamming code (single-error correcting)

\[ G = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \quad H = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix} \]

<table>
<thead>
<tr>
<th>Info vector</th>
<th>Code word</th>
<th>Error pattern</th>
<th>Syndrome</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>0000000</td>
<td>00000000</td>
<td>000</td>
</tr>
<tr>
<td>10000</td>
<td>1101000</td>
<td>11000000</td>
<td>000</td>
</tr>
<tr>
<td>01000</td>
<td>0110100</td>
<td>01000000</td>
<td>000</td>
</tr>
<tr>
<td>11000</td>
<td>1011100</td>
<td>10110000</td>
<td>000</td>
</tr>
<tr>
<td>00100</td>
<td>1110010</td>
<td>11100000</td>
<td>000</td>
</tr>
<tr>
<td>10100</td>
<td>0011101</td>
<td>00111010</td>
<td>000</td>
</tr>
<tr>
<td>01100</td>
<td>1000110</td>
<td>10001100</td>
<td>000</td>
</tr>
<tr>
<td>11100</td>
<td>0101110</td>
<td>01011100</td>
<td>000</td>
</tr>
<tr>
<td>00011</td>
<td>1010001</td>
<td>10100001</td>
<td>000</td>
</tr>
<tr>
<td>10011</td>
<td>0111001</td>
<td>01110001</td>
<td>000</td>
</tr>
<tr>
<td>01011</td>
<td>1100110</td>
<td>11001100</td>
<td>000</td>
</tr>
<tr>
<td>11011</td>
<td>0001110</td>
<td>00011100</td>
<td>000</td>
</tr>
<tr>
<td>00111</td>
<td>0100011</td>
<td>01000110</td>
<td>000</td>
</tr>
<tr>
<td>10111</td>
<td>1001111</td>
<td>10011110</td>
<td>000</td>
</tr>
<tr>
<td>01111</td>
<td>0010111</td>
<td>00101110</td>
<td>000</td>
</tr>
<tr>
<td>11111</td>
<td>1111111</td>
<td>11111110</td>
<td>000</td>
</tr>
</tbody>
</table>

Decoding example: Suppose \( \hat{u} = [0101] \).

Then the transmitting code word \( v = [1100101] \).

If the channel makes the 4th bit (from left) an error, \( r = [1101011] \).

Calculate syndrome \( s = [110] \). Look up table give \( \hat{e} = [0001000] \).

Get \( \hat{v} = r + \hat{e} = [1100101] \) \( \Rightarrow \hat{u} = [0101] \).