Drawing Eye Diagrams

There are many ways to draw eye diagrams. There is no right approach; it is subjective and personal. Dr Terman has outlined one technique during the lectures. Here we discuss some ideas. What you will be typically given:

- i) A unit-step response h[n], or the step response s[n].
- ii) The number of samples per bit N.

This is what you do.

Step 1: [Figure out the number B of interfering bits.] This can be done by looking at a diagram similar to Figure 1. Recall the "flip-and-slide" convolutional sum from lecture (or as I call it "filter-form"). Flip the unit sample response h[n], as shown. The number of whole bit cells covered by h[-n], plus 1, will be the number B of interfering bit cells.

If you want a formula¹ to calculate B, we can write

$$B = \left\lceil \frac{L-1}{N} \right\rceil + 1,$$

where L is the length (or "active-part") of h[n] (see Figure 1). Note that $\lceil a/b \rceil$ is the "ceiling" of the fraction a/b, i.e. the smallest integer that is larger than of equal to a/b. For example, if a/b = 5/6, then $\lceil 5/6 \rceil = 1$. If a/b is an integer, i.e. a/b = 2, then $\lceil 2 \rceil = 2$.

- Step 2: [Get the step response s[n]].
- Step 3: [Consider each bit pattern]. There all together 2^B bit patterns, because we care about B interfering bit cells, plus the bit-cell of interest, see Figure 1. For each length-B bit pattern, we build the sequence of ascending and descending step responses, shown in Figure 2. Do the following [for each length-B bit pattern.]:
 - a) Set k = B 1. Initialize the output sequence y[n] := 0 (i.e. to the all-0 sequence).

¹There is another formula $B = \lfloor \frac{L}{N} \rfloor + 2$ given in Lecture notes 5. Both will work just fine (though note they are not exactly the same - I leave it up to you to think why this is so [but not so important]).



Figure 1: How many bit cells B do we need to consider?



Figure 2: Each bit transition is associated with a step transition s[n]. If the bit transits from $0 \rightarrow 1$, it is an *ascending* transition. If bit transits $1 \rightarrow 0$, the transition is a *descending* one.

- b) Look at the bit-cell boundary at $n = -k \cdot N$.
 - If there is **no** bit transition between the adjacent bit-cells (that share common boundary at $n = -k \cdot N$), **do nothing**.
 - If there is an ascending transition $0 \rightarrow 1$, draw the ascending (shifted) step response

$$s[n+k\cdot N]$$

Set $y[n] := y[n] + s[n+k \cdot N].$

If there is an **descending** transition $1 \rightarrow 0$, draw² the **descending** (shifted) step response

$$-s[n+k\cdot N].$$

Set
$$y[n] := y[n] - s[n+k \cdot N].$$

c) Set k := k - 1. If k < 0 you are **done with this bit pattern**, and you will get the corresponding output sequence y[n]. Otherwise **repeat** step b).

The idea of draw the transitions is purely for visual effect. Its best to see how this works by examples.

Example 1. Let us consider Problem 5 of Tutorial "Noise & Bit Errors".

- Number of samples per bit N = 3.
- Step response (memory length L = 4)

$$s[n] = 0.2, 0.4, 0.7, 1.0, 1.0, \cdots$$

Step 1: We have $B = \lfloor (L-1)/N \rfloor + 1 = \lfloor 3/4 \rfloor + 1 = 2$.

Step 2: Already given.

Step 3: We need to consider bit patterns of length B = 2. There are a total of 4 bit patterns, as seen in Figure 3. The bit patterns with the corresponding output sequences are given in the following table.

Bit pattern	Output $y[n]$	Bit pattern	$Output \ y[n]$
0,0	0	1, 1	s[n+3]
0,1	s[n]	1,0	s[n+3] - s[n]

²I really draw it as $s[\infty] - s[n + k \cdot N]$, but this will not be important, you can come up with your own way. I only want to keep track which transitions are descending.



Figure 3: Example 1.

Example 2. Let us consider Problem 6 of Tutorial "LTI Systems, Intersymbol Interference, Deconvolution".

- Number of samples per bit N = 5.
- Step response (memory length L = 10)

 $s[n] = 0.0, 0.04, 0.12, 0.24, 0.40, 0.60, 0.72, 0.84, 0.96, 1.00, \cdots$

Step 1: We have B = [(L-1)/N] = [9/5] = 2.

Step 2: Already given.

Step 3: We need to consider bit patterns of length B = 3. There are a total of 8 bit patterns, as seen in Figure 4. The bit patterns with the corresponding output sequences are given in the following table.

Bit pattern	$Output \ y[n]$	Bit pattern	$Output \ y[n]$
0, 0, 0	0	1, 1, 1	s[n+10]
0, 0, 1	s[n]	1, 1, 0	s[n+10] - s[n]
0, 1, 0	s[n+5] - s[n]	1, 0, 1	s[n+10] - s[n+5] + s[n]
0, 1, 1	s[n+5]	1, 0, 0	s[n+10] - s[n+5]

Remark 1. You might have noticed a pattern from Examples 1 and 2. The two bit patterns in the same row of the tables, are flips of each other, e.g. in Example 1, the first row pattern 0,0 is a flipped version of 1,1. The outputs y[n] of these two bit patterns look like flips of each other too. If I denote the outputs of the left and right (flipped) bit patterns as y[n] and y'[n], respectively, then they satisfy the relation

$$y'[n] = s[n + (B - 1)N] - y[n]$$

This is really a geometric reflection about the "middle-point" s[n + (B-1)N].

³Recall that we are only interested in evaluating y[n] for time instants $n \ge 0$. Figure 1 actually gives quite a convincing argument that $s[n + (B-1)N] = s[\infty]$ for all $n \ge 0$. So the values s[n + (B-1)N] that we actually require, is really some constant $s[\infty]$. This observation further simplifies computations.



Figure 4: Example 2.

In Example 1, the bit pattern 0,1 has output y[n] = s[n], and the flipped bit pattern 1,0 has output y'[n] = s[n+3] - y[n] = s[n+3] - s[n].

This means that we really only need to determine the left bit patterns; the right (flipped) patterns are obtained automatically by flipping the left outputs. This reduces the number of patterns we need to consider by half.

Remark 2. If you get more used to eye diagrams, you don't even need to draw stuff in step b). From the tables, there is an observable relationship between the bit patterns, and the outputs y[n]. With practice, one could directly build the tables very efficiently, and this could be well-suited for quizzes.

Superimposing all the outputs y[n] on top of each other, will give the eye diagram.



Here I only label the left bit patterns.

Another way to generate eye diagrams

Prof. Shah suggested an interesting approach to get eye diagrams. This is done using special sequences known as *deBruijn sequences*. A deBruijn sequence is really a code (something similar to 8b/10b), and has the following properties

- It is a binary sequence.
- It has a parameter m, which determines its length 2^m .
- If we periodically replicate the sequences, every possible length-*m* bit pattern appears.

Example 3. The deBruijn sequences of lengths $2^m = 4, 8, 16$ and 32 are

m	Sequence	Unit Steps (N samples per bit)
2	$0, 0, 1, 1 \ (, 0, 0, 1, \cdots)$	u[n-2N] - u[n-4N]
3	$0, 0, 0, 1, 1, 1, 0, 1 \ (, 0, 0, 0, 1, \cdots)$	u[n-3N] - u[n-6N] + u[n-7N]
		-u[n-8N]
4	$0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1$ $(, 0, 0, 0, 0, 1, \cdots)$	u[n-4N] - u[n-8N] + u[n-9N]
		-u[n-10N] + u[n-11N] - u[n-13N]
		+u[n-15N] - u[n-16N]
5	omitted because I don't want to write	$(transition \ pts \ (multiples \ of \ N))$
	a string of 32 bits	4, 5, 6, 7, 8, 11, 12, 14,
		17, 22, 24, 26, 27, 28, 30, 31

For the sequence of length $2^m = 4$, the first two bits are 0,0, the next two 0,1, the next two 1,1, and the next two 1,0. Similarly for the sequence of length $2^m = 8$, every length-3 bit pattern (e.g. 0,0,0, and 0,0,1, and 0,1,0, etc) will appear. The length-4 bit patterns appear in the last sequence of length $2^m = 16$. You get the idea.

We can use deBruijn sequences to generate eye diagrams as follows:

Step 1: Get the deBruijn sequence of length $2^m = 2^B$.

Step 2: Transmit this deBruijn sequence (using N samples per bit) and get channel output y[n].

Step 3: Each bit-cell now contains the output corresponding to the length-B bit pattern.

Example 4. The figures below show how we use deBruijn sequences to get all possible outputs y[n] for previous Examples 1 and 2. The nice thing here is that we only need 2 transitions for Example 1, and 4 transitions for Example 2. These are small numbers of transitions.



The formula I know for generating these sequences, is non-trivial to evaluate in a quiz setting. Still you have a crib-sheet for the quiz, and you can copy down the deBruijn sequences if you want to use them. They are compactly represented using unit-steps. Hopefully I did not make any mistakes when compiling these sequences.