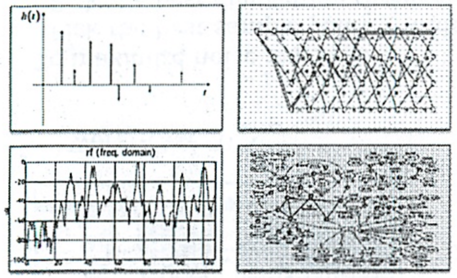
2/16



INTRODUCTION TO EECS II

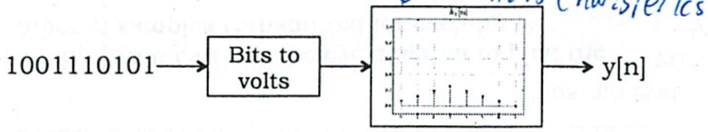
DIGITAL COMMUNICATION SYSTEMS

6.02 Spring 2011
Lecture #5

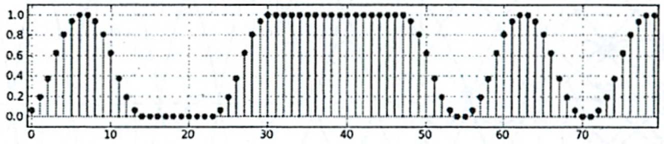
- Intersymbol interference
- Deconvolution
- Stability & noise, approximate deconvolvers

Transmission Over a Channel

← Channels charisteics

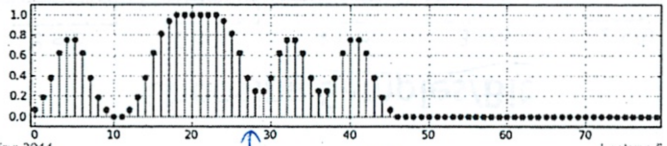


$x[n] * h_4[n]$ at 8 samples/bit



Sample

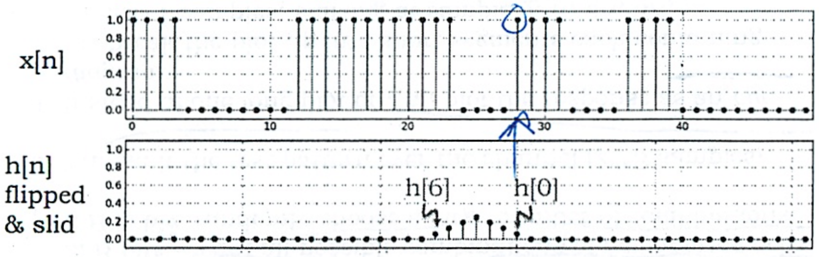
$x[n] * h_4[n]$ at 4 samples/bit



look at 28
looks like worse case 0

same as 6.041

Convolution sum: "flip and slide"



y[28] = x[28]h[0] + x[27]h[1] + … + x[22]h[6] ↑ prior 7 inputs matter

← for one output

Visual representation of convolution sum: do a horizontal flip of the of graph of h[n], then slide along under x[n].

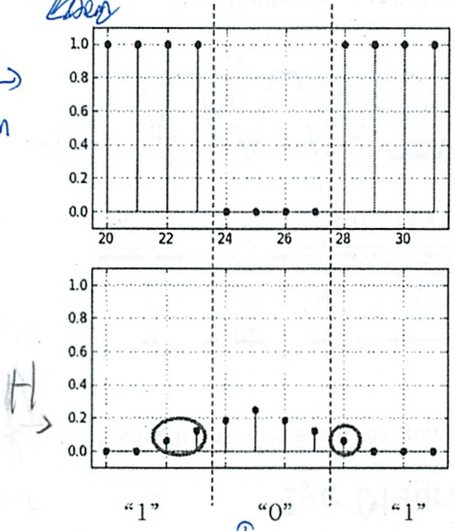
To compute y[m], slide flipped h[n] until h[0] is under x[m], then compute sum of element-by-element product of the two sequences.

Zoom

zoom in

Intersymbol Interference (ISI)

transmition spread out over time



Issue:
If we send a small number of samples/bit, the active portion of h[n] may cover more than one bit cell when doing convolution sum.

Result:
y[n] values for a particular bit cell include contributions from neighboring cells.

Example: y[28] is the lowest voltage received for the "0" bit, but includes contributions from the neighboring "1" bits.

↑ largest h values line up w/ 0 – which is correct – but some neighboring's are effective

2/16

Strategy:

Given h[n], how bad is ISI?

In general - our H covers 3 bit cells - so look at all possible combos of 3 bits

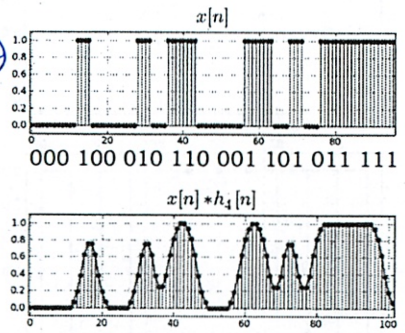
Recipe:

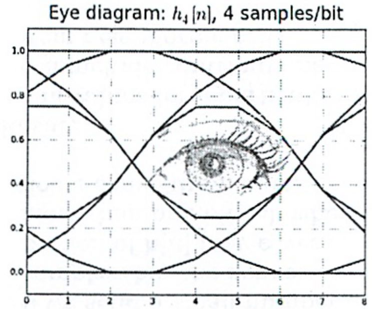
1. Compute B, the number bits "covered" by h[n]. Let N = samples/bit
$$B = \left\lfloor \frac{\text{length of active portion of h[n]}}{N} \right\rfloor + 2$$
2. Generate a test pattern that contains all possible combinations of B bits – want all possible combinations of neighboring cells. If B is big, randomly choose a large number of combinations.
3. Transmit the test pattern over the channel (2^N*B samples)
4. Instead of one long plot of y[n], plot the response as an *eye diagram:*
   a. break the plot up into short segments each containing 2N+1 samples, starting at sample 0, N, 2N, 3N, …
   b. plot all the short segments on top of each other

Output

Eye Diagram Example

Using h_4[n] and samples_per_bit=4: N = 3





Eye diagrams make it easy to find the worst-case signaling conditions at the receiving end.

instead of looking through music for a C#
Plot all measures on top of each other and look for note

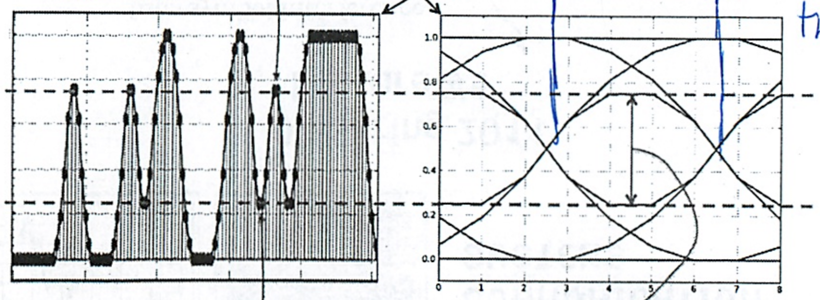
"Width" of Eye

Both plots have same info

Both plots show the same y[n]

bitcell N - sample in the middle

Worst-case "1"



Worst-case "0"

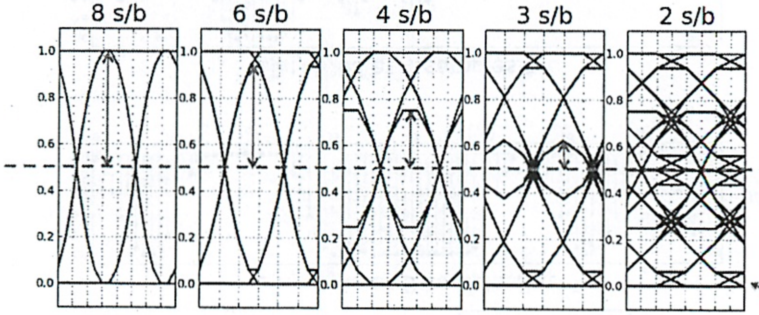
2 bins so see eye

"width" of eye
(as in "eye wide open")

well "height" - but called width in EECS

To maximize noise margins:
Pick the best sample point → widest point in the eye
Pick the best digitization threshold → half-way across width

Choosing Samples/Bit



Oops, no eye!

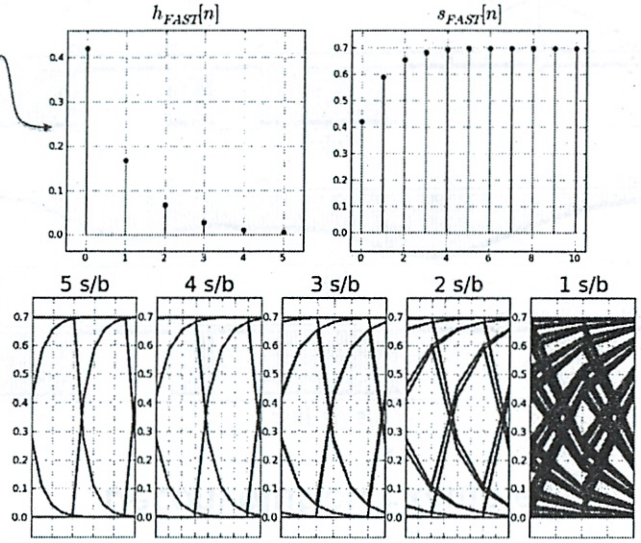
no obvious transitions

Given h[n], you can use the eye diagram to pick the number of samples transmitted for each bit (N):

Reduce N until you reach the noise margin you feel is the minimum acceptable value.

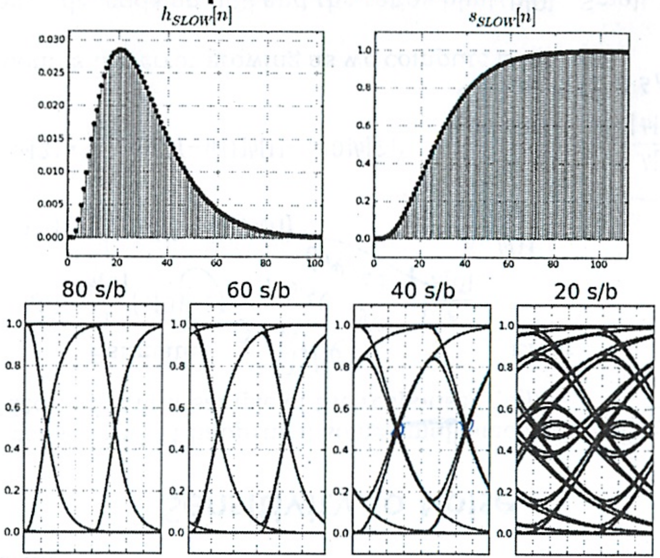
Example: “fast” channel

h[n] for actual for actual channels extend to +∞, although values get vanishing small. For practicality, we truncate when terms fall below a certain value.



still a tiny eye but Chris would not buy it

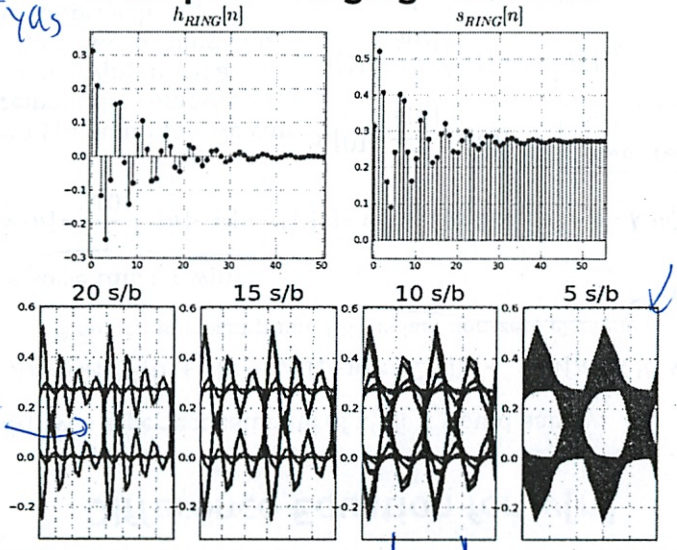
Example: “very slow channel”



no eye here all transitions should cross

Example: “ringing” channel

have gotyas



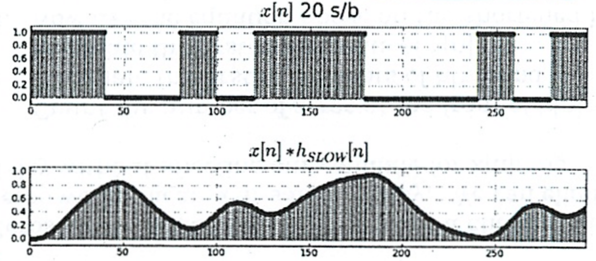
Actually much better at the natural freq

- but longer cable rings differently

1 bit cell

don't sample in middle

Can We Recover From ISI?



After all, in a perfect world (no noise), no information has been lost, only spread out over many samples.

Given y[n] and h[n], can we develop an estimate w[n] for the actual input waveform x[n]? We could, of course, easily receive x[n]!

de convolution

Difference Equation for w[n]

If w[n] was a perfect estimate of x[n], it would satisfy:

$$y[n] = w[n]h[0] + w[n-1]h[1] + w[n-2]h[2] + \ldots + w[n-K]h[K]$$

Simplifying assumption: h[K] is last non-zero element

finite lenght

Let's solve this for w[n]:

$$w[n] = \frac{1}{h[0]}\left(y[n] - w[n-1]h[1] + w[n-2]h[2] + \ldots + w[n-K]h[K]\right)$$

$h[i]=0$	$i<0$ or $i>K$
$w[j]=0$	$j<0$

Given y[n] and h[n], we can incrementally compute sequence w[n] using a straightforward "plug and chug" approach:

$$w[0] = \frac{1}{h[0]}(y[0])$$

$$w[1] = \frac{1}{h[0]}(y[1] - w[0]h[1])$$

$$w[2] = \frac{1}{h[0]}(y[2] - w[1]h[1] - w[0]h[2])$$

Computers do this very well

What if h[0]=0?

$$w[n] = \frac{1}{h[0]}\left(y[n] - w[n-1]h[1] + w[n-2]h[2] + \ldots + w[n-K]h[K]\right)$$

Oops! Division by 0 isn't a good idea...

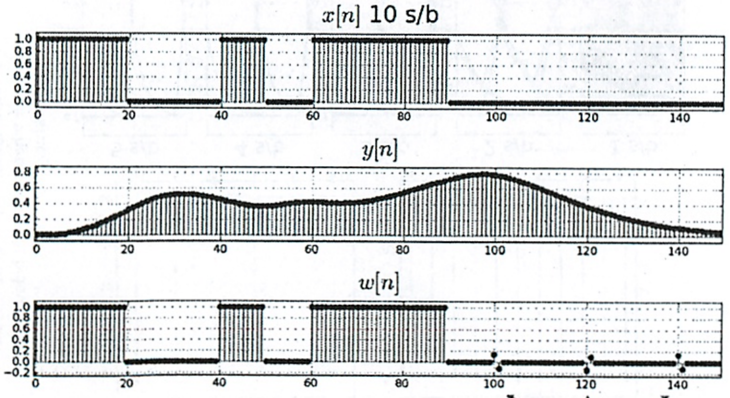
Zeros at the beginning h[n] represent a channel with a delay: m zeros would mean a m-sample delay. We can eliminate the delay without affecting our estimate for x[n]. So

1. Count the number of zeros at the front of h[n] = m
2. Eliminate the first m elements of h[n], and eliminate the first m elements of y[n]
3. Now use the equation above on the shortened h[n] and y[n]

h[0) means 1 sample delay

So count; shift; get rid of it

Deconvolution Example



???
(hint: see slide #10)

truncated at 100

So deconvoler does not like missing infatesimally small values

Sensitivity to Noise

Let's consider what happens if some small amount of noise (ε) is added to the first sample of the response (y[0]):

Estimate	Error
$w[0] = \frac{1}{h[0]}(y[0] + \varepsilon)$	$\frac{\varepsilon}{h[0]}$
$w[1] = \frac{1}{h[0]}(y[1] - w[0]h[1])$	$-\frac{\varepsilon}{h[0]}\frac{h[1]}{h[0]}$
$w[2] = \frac{1}{h[0]}(y[2] - w[1]h[1] - w[0]h[2])$	$-\frac{\varepsilon}{h[0]}\left(\frac{h[1]}{h[0]}\right)^2 - \frac{\varepsilon}{h[0]}\frac{h[2]}{h[0]}$

a little noise on one sample

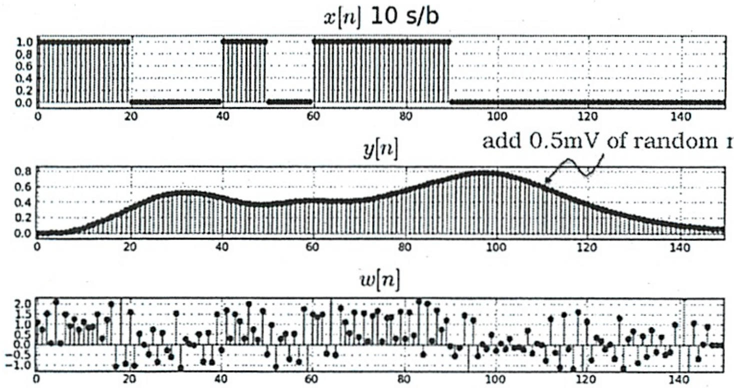
when h(0) is small

or when ratios are bigger than 1

Question: is the error growing as we compute more w's?

Answer: depends on h[0] and the ratios h[m]/h[0]. Small values of h[0] and (h[m]/h[0]) > 1 are troublesome...

Noisy Deconvolution Example



add 0.5mV of random noise

de convolution screws up big time

Urk!

Stability Criterion

Conservative Criteria

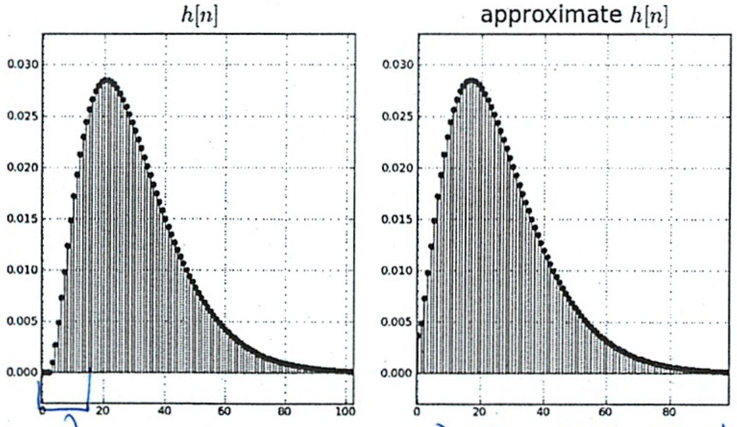
The notes have a derivation of the following sufficient (very conservative) condition that will ensure the stability of the deconvolver operating on a noisy y[n]:

$$\sum_{m=1}^{K}\left|\frac{h[m]}{h[0]}\right|<1 \quad \text{or, perhaps more usefully} \quad \sum_{m=1}^{K}|h[m]|<|h[0]|$$

What if my h[n] doesn't meet this criterion?

Make a new "approximate" h[n] that does! Combine samples at the beginning of h[n] to make a bigger h[0].

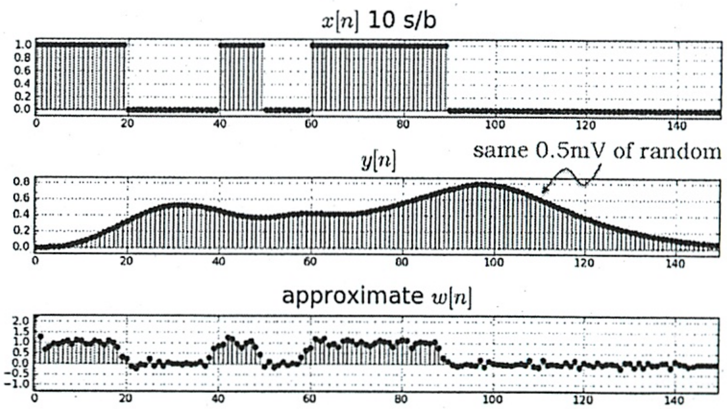
Example Approximate $h_{SLOW}[n]$



new approx sample

Approximation: combine first 5 samples of $h_{SLOW}[n]$

(Less) Noisy Deconvolution Example



same 0.5mV of random noise

not turning all that bad due to noise

mis estimated start of transitions

To save your work, click the SAVE button at the bottom of this page. You can revisit this page, revise your answers and SAVE as often as you like.

To submit the assignment, click the SUBMIT button at the bottom of this page. YOU CAN SUBMIT ONLY ONCE. Once the assignment has been submitted, you can continue to view this page but will no longer be able to make any changes to your answers.

6.02 Spring 2011: Plasmeier,Michael E.

PSet PS2

Dates & Deadlines

issued:	Feb-09-2011 at 00:00
due:	Feb-17-2011 at 06:00 (Feb-22-2011 at 06:00 with extension)
checkoff due:	Feb-22-2011 at 06:00

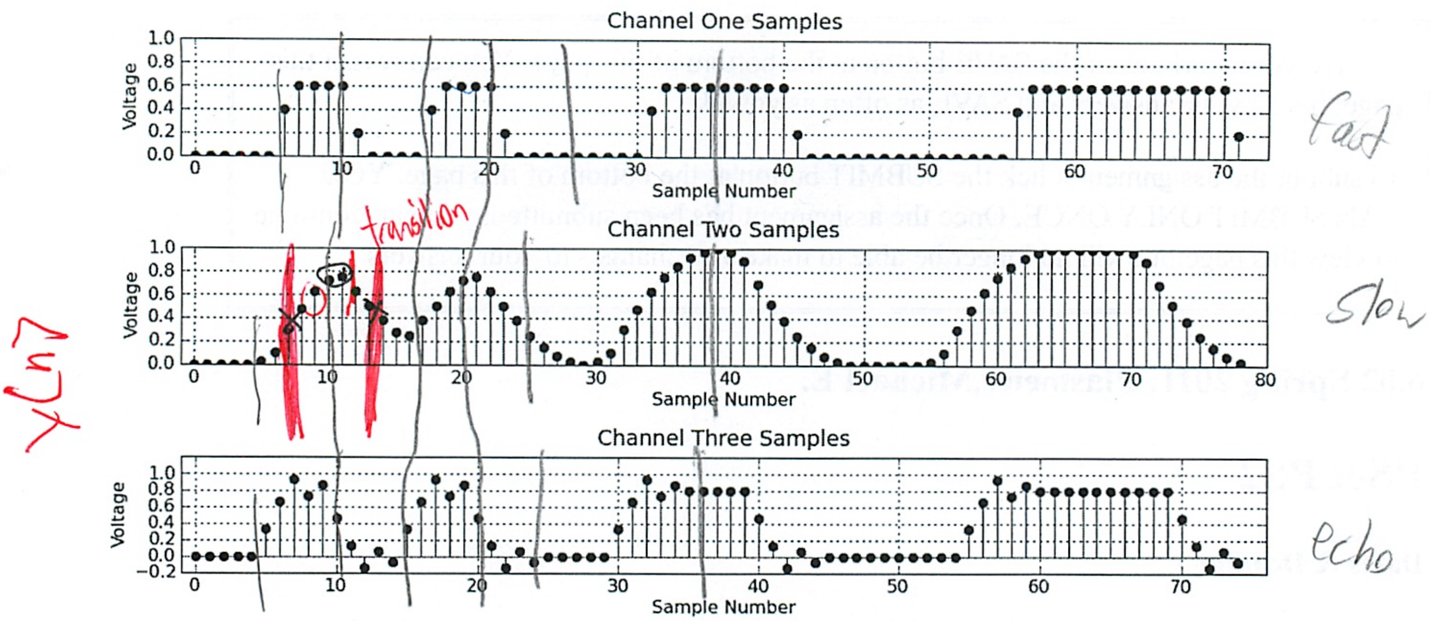
Help is available from the staff in the 6.02 lab (38-530) during lab hours -- for the staffing schedule please see the Lab Hours page on the course website. We recommend coming to the lab if you want help debugging your code.

For other questions, please try the 6.02 on-line Q&A forum at Piazzza.

Your answers will be graded by actual human beings, so your answers aren't limited to machine-gradable responses. Some of the questions ask for explanations and it's always good to provide a short explanation of your answer.

Problem 1. Digitization Thresholds (3 points)

Consider three different channels (each of which happen to be linear and time-invariant, though you will not really need that fact to answer this question). The three channels are denoted, perhaps unoriginally, as *channel one*, *channel two*, and *channel three*. The input to each of the channels is a voltage sample sequence associated with transmitting bits using five voltage samples per bit, and the channel output voltage samples are shown in the three plots below.



Note: the voltage samples of channel one's response never rise above 0.6 volts; there are several cases where channel three's response is ~0.8 volts for several samples in a row. Please use these plots to answer all the parts of this question, and please keep in mind that *five voltage samples are used to represent each bit.*

A. What theshold voltage should be used for each of the channels?

? same for each

Channel one .3 V
two .4 V
three .4

(points: 1)

B. What 14-bit sequence is being transmitted (the sequence is the same for each channel)? Enter your answer as a sequence of 0's and 1's.

01010011000111

(points: 0.5)

C. For each of the channel output sequences, please select a good set of five sample indices to use for detecting the first 5 received bits. For this case of a short sequence of bits, please assume that the distance between bit detection sample indices is equal to the number of samples per bit (which is five in this example). Please note that for each channel, there are several sets of bit detection sample indices that will lead to correct bit identification. However, it is a good strategy to use bit detection samples that are half-way between potential rising or falling transitions between bits (as seen at the

what does this mean? wrote in

receiver). That way, even if there are small timing errors, bits will stil be identified correctly.

6h – middle, etc

Appropriate sample indicies for Channel 1:

3

(points: 0.5)

Appropriate sample indicies for Channel 2:

3 Slight delay in channel
reciever needs to find transmit clock

(points: 0.5)

~~For~~ Algo can have non causal -
does not matter where true transition is
just want cleanest point

Appropriate sample indicies for Channel 3:

3

~~Have~~ can when receiving -> transitions
when it crosses the threshold

(points: 0.5)

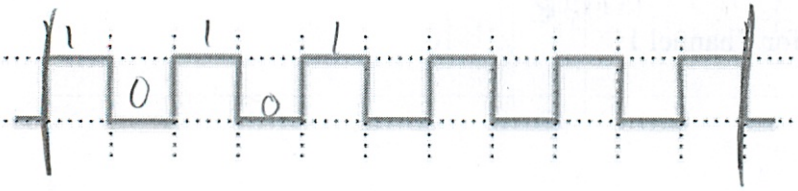
Problem 2. Clock recovery (3 points)

The symbol rate of a transmitter specifies how quickly successive symbols of the message are transmitted. For example, if the symbol rate is 1,000,000 symbols/second, each symbol would be transmitted for 1 microsecond. Typically the symbol rate is fixed.

The receiver can deduce some information about the transmitter's symbol rate from the incoming waveform. Specifically, each transition in the waveform marks the start of a new symbol -- so each transition of the incoming waveform adds another clue to what the symbol rate must be.

In the following plots of received waveforms, the transmitter is sending sequences of binary symbols (i.e., either 0 or 1) at some fixed symbol rate, using 0V to represent 0 and 1V to represent 1. For each of the waveforms, indicate the slowest possible symbol rate that's consistent with the transitions in the waveform. The horizontal grid spacing is 1 microsecond (1e-6 sec). Your answer should be a number with the units of symbols/second. Using that symbol rate, decode the waveform into a sequence of 0s and 1s; ignore partial symbol transmissions (if any) at the beginning and end of the plot.

A.



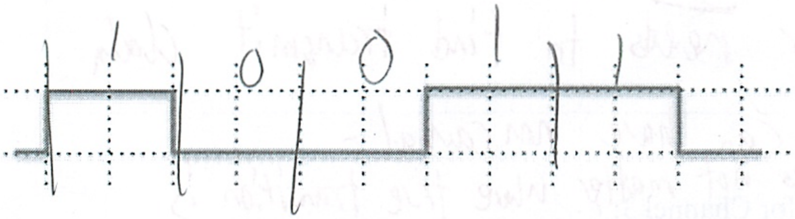
Slowest symbol rate and decoded bit string:

1,000,000 bit/sec

1010101010101

(points: 1)

B.



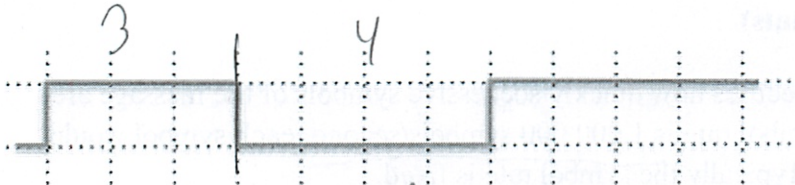
Slowest symbol rate and decoded bit string:

500,000 symbol/sec

10011

(points: 1)

C.



Must be 1 - only common factor of 3 and 4

Slowest symbol rate and decoded bit string:

11100001111

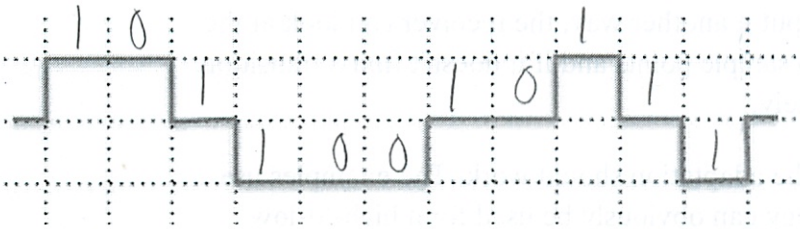
(points: 1)

Problem 3. Differential encoding (1 point)

Some versions of Ethernet use an *MLT-3* (Multi-Level Transmit) encoding that transmits one of three voltage levels (-1, 0, +1). MLT-3 cycles through the voltage levels in the sequence -1, 0, +1, 0. To transmit a "1", the sender changes the voltage at the beginning of the clock cycle; to transmit a "0" it maintains the same voltage. Thus the information to be transmitted is encoded by differences (or lack thereof) in the voltage, not by the absolute voltage level itself. For example the +1 voltage may represent either a "0" or a "1" depending on whether the voltage changed at the beginning of the clock cycle or not. Using differences to transmit information rather than levels is called *differential encoding*.

More information can be found in the MLT-3 Wikipedia article. We encourage you to use the web to read up on the various technologies we mention in 6.02.

Please list the first 10 bits that can be decoded from the following MLT-3 waveform. The vertical grid lines show the beginning of each clock cycle.



How does it start?

First decoded 10 bits:

~~1~~0 1100 10 111

~~? disregard?~~

don't disregard

(points: 1)

Send email if marked wrong

Python Task 1. Clock and data recovery (8 points)

Useful download links:

PS2_tests.py -- test jigs for this assignment
PS2_1.py -- template file for this task

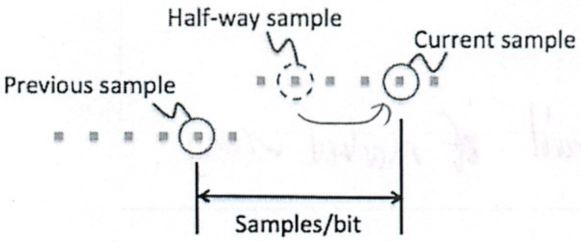
Your goal for this task is write a Python procedure that processes a sequence of received voltage samples to produce a sequence of received bits. The procedure is given `samples_per_bit`, the number of voltage samples the transmitter generated for each bit:

In a perfect world, it would be a trivial task to find the voltage sample in the middle of each bit transmission and use that to determine the transmitted bit. Just start the sampling index at `samples_per_bit/2`, then increase the index by `samples_per_bit` to move to the next voltage sample, and so on until you run out of voltage samples.

Alas, in the real world things are a bit more complicated. Both the transmitter and receiver use an internal clock oscillator running at the sample rate to determine when to generate or acquire the next voltage sample. And they both use counters to keep track of how many samples there are in each bit. The complication is that the frequencies of the transmitter's and receiver's clock may not be exactly matched. Say the transmitter is sending 5 voltage samples per message bit. If the receiver's clock is a little slower, the transmitter will seem to be transmitting faster, e.g., transmitting at 4.999 samples per bit. Similarly, if the receiver's clock is a little faster, the transmitter will seem to be transmitting slower, e.g., transmitting at 5.001 samples per bit. This small difference accummulates over time, so if the receiver uses a static sampling strategy like the one outlined in the previous paragraph, it will eventually be sampling right at the transition points between two bits. And to add insult to injury, the difference in the two clock frequencies will change over time.

The fix is to have the receiver adapt the timing of it's sampling based on where it detects transitions in the voltage samples. The transition (when there is one) should happen half-way between the chosen sample points. Or to put it another way, the receiver can look at the voltage sample half-way between the two sample points and if it doesn't find a transition, it should adjust the sample index appropriately.

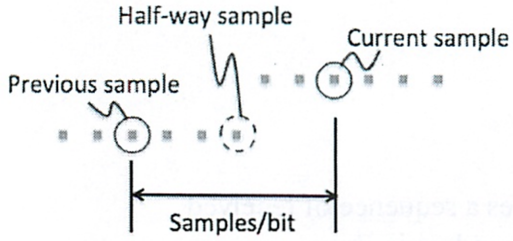
The following two figures illustrate how the adaptation should work. The examples use a low-to-high transition, but the same strategy can obviously be used for a high-to-low transition. The two cases differ in value of the sample that's half-way between the current sample point and the previous sample point. Note that a transition has occurred when two consecutive sample points represent different bit values.



Case 1: the half-way sample is the *same* as the current sample. In this case the half-way sample is in the same bit transmission as the current sample, i.e., we're sampling too late in the bit transmission. So when moving to the next sample, increment the index by `samples_per_bit - 1` to move "back".

is this the index ↓ or i

make this dynamically longer/shorter – think just index!



Case 2: the half-way sample is *different* than the current sample. In this case the half-way sample is in the previous bit transmission from the current sample, i.e., we're sampling too early in the bit transmission. So when moving to the next sample, increment the index by `samples_per_bit + 1` to move "forward".

but how to always know – think I remember update cont'easly

If there is no transition, simply increment the sample index by `samples_per_bit` to move to the next sample. This keeps the sampling position approximately right until the next transition provides the information necessary to make the appropriate adjustment.

If you think about it, when there is a transition, one of the two cases above will be true and so we'll be constantly adjusting the relative position of the sampling index. That's fine -- if the relative position is close to correct, we'll make the opposite adjustment next time. But if a large correction is necessary, it will take several transitions for the correction to happen. To facilitate this initial correction, in most protocols the transmission of message begins with a *training sequence* of alternating 0- and 1-bits (remember each bit is actually `samples_per_bit` voltage samples long). This provides many transitions for the receiver's adaptation circuity to chew on.

Please write a Python procedure `data_recovery` that takes a sequence of digitized voltage samples (i.e., voltage samples that have already been compared against a threshold and converted to "0" or "1") and returns a sequence of received bits. The procedure is also given `samples_per_bit.`

PS2_1.py is a template file for this task. The calls to `PS2_tests.task1_test(...)` invokes your data recovery function on several different digitized bit sequences. The first one is "perfect" in the sense that the receiver and transmitter clocks have exactly the same frequency and phase. The last two test sequences are more challenging, with the two clocks slowly drifting with respect to each other. To successfully recover the message bits your code must continually use the adapatation procedure described above.

```
# PS2_1.py -- template for task #1
import PS2_tests

# this routine simply samples periodically and does not
# do adaptation using the transitions.
def naive_data_recovery(digitized_samples,samples_per_bit):
    l = len(digitized_samples)
    result = []  # accumulate message bits here

    # start in middle of first message bit
    index = samples_per_bit/2

    # iterate through samples until end
    while index < l:
        current = digitized_samples[index]
        result.append(current)
        # move to middle of next bit
        index += samples_per_bit

    return result

# return sequence of message bits given sequence of received
# digitized samples and the number of samples transmitted
# for each bit.  Handle the case where the receiver's clock
# is slightly faster or slower than the transmitter's.
def data_recovery(digitized_samples,samples_per_bit):
    # **** YOUR CODE HERE ****
    return []

# testing code.  Do it this way so we can import this file
# and use its functions without also running the test code.
if __name__ == '__main__':
```

So works for first

← is in the middle

not same as current on paper?
look at tests

but ∞ loop
||

first attempt much worse - should it always move?
moving too much - did I follow instructions?

✱ Only if transition!

Keeps moving up too much

```
# which function to test
ftest = naive_data_recovery
#ftest = data_recovery

# start with a short test, no clock drift
PS2_tests.task1_test(ftest,'same',debug=True,nbits=16)

# clocks are drifting
PS2_tests.task1_test(ftest,'fast')
PS2_tests.task1_test(ftest,'slow')
```

! Oh samples-per-bit -1 since 0 indexed!

passed first - but not others!

When you're ready, please submit the file with your code using the field below.

File to upload for Task 1: Browse...

Most should be no trans

(points: 8)

Python Task 2. 8b/10b decoding (11 points)

training goes well

Useful download links:

PS2_2.py -- template file for this task

0111 1111

This task investigates a digital signaling protocol that is used by many high-speed digital transmission systems (PCIe, Firewire, USB 3.0, SATA, ...). This protocol was developed to help address the following issues:

- If the transmitter is sending bits continuously and the receiver starts listening at some point in the transmission, there's no way to locate the start of multi-bit symbols unless there's a long pause in the transmission that the receiver can interpret as "no data" and thus synchronize with the data stream.
- For electrical reasons it's desirable to maintain DC balance on the wire, i.e., that on the average the number of 0's is equal to the number of 1's.
- Transitions in the received bits indicate the start of a new bit and hence are useful in synchronizing the sampling process at the receiver -- the better the synchronization, the faster the maximum possible symbol rate. So ideally one would like to have frequent transitions. On the other hand each transition consumes power, so it would be nice to minimize the number of transitions consistent with the synchronization constraint and, of course, the need to send actual data! In a signaling protocol where the transitions are determined by the message content may not achieve these goals.

here it seems I messed up +/- - but followed spec

Others never broke since no ISI in it

too few bits now

Passed!

To address these issues we can use an *encoder* at the transmitter to recode the message bits into a sequence that has the properties we want, and use a *decoder* at the receiver to recover the original message bits. Many of today's high-speed data links (e.g., PCI-e and SATA) use an 8b/10b encoding scheme developed at IBM. The 8b/10b encoder converts 8-bit message symbols into 10 transmitted bits. There are 256 possible 8-bit words and 1024 possible 10-bit transmit symbols, so one can choose the mapping from 8-bit to 10-bit so that the the 10-bit transmit symbols have the following properties:

the Js are really there - must be break or something

- the maximum run of 0's or 1's is five bits (i.e., there is at least one transition every five bits).
- at any given sample the maximum difference between the number of 1's received and the number of 0's received is six.
- special 7-bit sequences can be inserted into the transmission that don't appear in any consecutive sequence of encoded message bits, even when considering sequences that span two transmit symbols. The receiver can do a bit-by-bit search for these unique patterns in the incoming stream and then know how the 10-bit sequences are aligned in the incoming stream.

Here's how the encoder works: collections of 8-bit words are broken into small groups of words (16 words/group in this task) called a *packet*. The last packet is padded with NULLs if the message doesn't happen to be an exact multiple of 16 symbols. Each packet is sent using the following wire protocol:

- A sequence of alternating 0 bits and 1 bits are sent first (recall that each bit is multiple voltage samples). This sequence is useful for making sure the receiver's clock recovery machinery has synchronized with the transmitter's clock. These bits aren't part of the message; they're there just to aid in clock recovery.
- A 7-bit SYNC pattern -- either 0011111 or 1100000 where the least-significant bit (LSB) is shown on the left -- is transmitted so that the receiver can find the beginning of the packet. **Note that the SYNC patterns are transmitted least-significant bit (LSB) first.** C,
- The sixteen 10-bit transmit symbols are sent -- the *packet data*. Each 10-bit transmit symbol is determined by table lookup using the 8-bit word as the index. **Note that all 10-bit symbols are transmitted least-significant bit (LSB) first.** 16 bits at once

Multiple packets are sent until the complete message has been transmitted. Note that there's no particular specification of what happens between packets -- the next packet may follow immediately, or the transmitter may sit idle for a while, sending, say, training sequence samples. - at new packet

oh good don't need to build

Write a Python procedure `receive` that takes a single argument -- a sequence of bits such as might be output by your code in Task #1 -- and returns the sequence of characters that were contained in the packet(s). Here's how to proceed:

- To determine where a packet starts in the received bit stream, look for instances of the SYNC patterns. You'll have to search bit-by-bit to detect where in the bit stream the packet starts. Once you've located an instance of the SYNC patterns, the next bit

↓ not right most

following the pattern is the LSB of the first 10-bit transmit symbol which encodes the first 8-bit message symbol in the packet. Don't forget that the SYNC patterns are appearing LSB first in the received bit stream.

- To decode each of the 16 data symbols, use `PS2_tests.bits_to_int` to convert each 10-bit sequence into an integer (first bit of the sequence is the least-significant bit of the integer) and use it as index into the list `lab1.table_10b_8b` to retrieve the integer representation of the original 8-bit message symbol. If the table entry contains `None` just ignore that particular 10-bit symbol, otherwise use Python's built-in `chr()` function to convert the 8-bit integer into a character which can be appended to a list that is accummulating all the received characters.

PS-2

How to convert to integer

- Once 16 data symbols have been decoded, the packet processing is done, so restart your search for a SYNC pattern, looking for the start of the next packet.

LSB = right most bit

- The number of message bytes being transmitted is not necessarily multiple of 16. Please trim off the NULL bytes that might appear at the end of the last packet.

PS2_2.py is a template file for this task. The call to `PS2_tests.task2_test(...)` invokes your receive function with an array of bits resulting from encoding a given message using an 8b/10b encoder.

totally wrong! Should be 'test'

```
# PS2_2.py -- template for task #2
import PS2_tests

# These are the two 7-bit sync patterns, LSB first
sync1 = [0,0,1,1,1,1,1]
sync2 = [1,1,0,0,0,0,0]

def receive_8b10b(received_bits,packet_size=16):
    """
    Convert a sequence of bits transmitted by a 8b/10b encoder into a
    sequence of message bytes.  The received bit sequence is made up
    of 16-byte packets preceded by one of the two 8b10b SYNC
    sequences.  There may be other bits between packets (e.g., bits
    serving as a clock training sequence) -- these should be ignored
    by your function.
    """

    # **** YOUR CODE HERE ****

    return []


# testing code.  Do it this way so we can import this file
# and use its functions without also running the test code.
if __name__ == '__main__':
    # short message is just the word "test"
    PS2_tests.task2_test(receive_8b10b,PS2_tests.short_message)
```

Oh I did wrong index - worked!

Now need to do new packet function

How to collapse list to string? str?

but getting lots of Js

just reject

```
    # now try a longer message with multiple packets
    PS2_tests.task2_test(receive_8b10b,PS2_tests.long_message)
```

When you're ready, please submit the file with your code using the field below.

File to upload for Task 2: Browse... ✓

(points: 8)

A. If the channel can support, say, a transmission rate of four million samples/second, what's the rate at which 8-bit symbols are produced by the receiver?

bits

~~well 10~~ Look at actual $\frac{1484}{17537} = 8.4\%$

(points: 1)

B. What are the pros and cons of increasing the frequency at which sync patterns (the special 7-bit sequences discussed in step 3 above) are embedded in the transmit stream? One factor to consider: how does the rate at which sync patterns are inserted affect your answer to the above question?

* remember diff layers!

(points: 1)

C. Suppose that a burst of noise corrupts the bit stream so that one of the bits is received incorrectly, e.g., a bit transmitted as 0 is received as 1. What effect does this have on the decoding process? Consider corrupted message bits separately from corrupted sync patterns bits.

(points: 1)

✓ Submitted 2/16

You can save your work at any time by clicking the Save button below. You can revisit this page, revise your answers and SAVE as often as you like.

Save

To submit the assignment, click on the Submit button below. YOU CAN SUBMIT ONLY ONCE after which you will not be able to make any further changes to your answers. Once an assignment is submitted, solutions will be visible after the due date and the graders will have access to your answers. When the grading is complete, points and grader comments will be shown on this page.

Submit

P-Set 2 Checkoff

2/17

- Also misread threshhold voltage
- Sample indicies

So the J was [0,1,0,1,0,1,0,1,0,1]

What pattern is that ???

Every 16

Index is a bit wrong

So it was

Words = 0

Words ≤ ~~16~~ 15

should have been

was getting junk after

Can use for loop

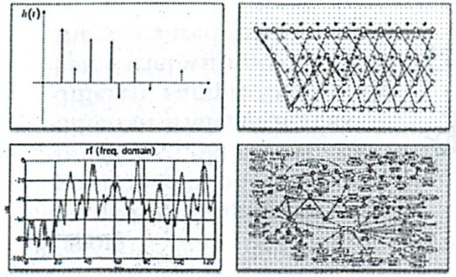
for i = [0:16]

for a in range [0:15]

Recitation

2/17

Absent



2/22

INTRODUCTION TO EECS II

DIGITAL COMMUNICATION SYSTEMS

6.02 Spring 2011
Lecture #6

- Mean, power, energy, SNR
- Metrics for random processes
- Normal PDF, CDF
- Calculating p(error), BER vs. SNR

Bad Things Happen to Good Signals

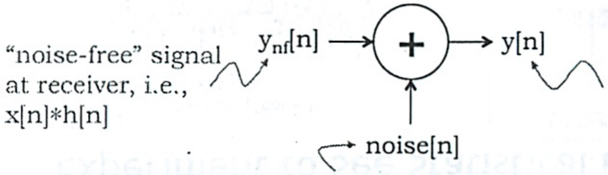
Noise, broadly construed, is any change to the signal from its expected value, x[n]*h[n], when it arrives at the receiver.

temp, voltage

We'll look at *additive noise* and assume the noise in our systems is independent in value and timing from the nominal signal, $y_{nf}[n]$, and that the noise can be described by a random variable with a known probability distribution.

for the given channel

We'll model the received signal as $y_{nf}[n]$ + noise[n].

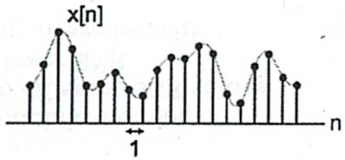

"noise-free" signal at receiver, i.e., x[n]*h[n]
$y_{nf}[n]$
+
y[n]
"noisy" signal receiver must process
noise[n]

Independent random noise

Seperate: Interfearence - not independent of 0 or 1 signal
- like Analog TV multi-path

6.011

Definition of Mean, Power, Energy



Slides 3-16 derived from 6.02 slides by Mike Perrott

Some interesting statistical metrics for x[n]:

Mean: $\mu_x = \frac{1}{N}\sum_{n=1}^{N} x[n]$ DC avg - want variations from mean

avg value

Power: $P_x = \frac{1}{N}\sum_{n=1}^{N} x[n]^2$ $\tilde{P}_x = \frac{1}{N}\sum_{n=1}^{N}(x[n]-\mu_x)^2$ subtract out mean

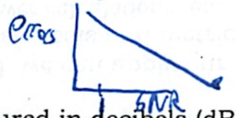
Energy: $E_x = \sum_{n=1}^{N} x[n]^2$ $\tilde{E}_x = \sum_{n=1}^{N}(x[n]-\mu_x)^2$

In analyzing our systems, we often use metrics where the mean has been factored out.

Signal-to-Noise Ratio (SNR)

The Signal-to-Noise ratio (SNR) is useful in judging the impact of noise on system performance: how bit error rate changes

$$\text{SNR} = \frac{\tilde{P}_{signal}}{\tilde{P}_{noise}}$$



SNR is often measured in decibels (dB):

$$\text{SNR (db)} = 10\log\left(\frac{\tilde{P}_{signal}}{\tilde{P}_{noise}}\right)$$

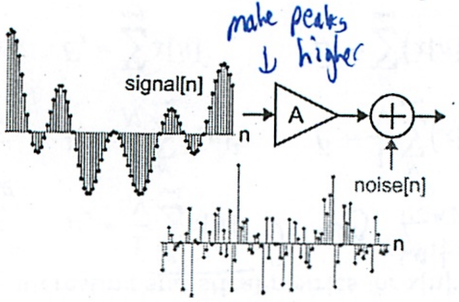
Many orders of magnitude

10logX	X
100	10000000000
90	1000000000
80	100000000
70	10000000
60	1000000
50	100000
40	10000
30	1000
20	100
10	10
0	1
-10	0.1
-20	0.01
-30	0.001
-40	0.0001
-50	0.000001
-60	0.0000001
-70	0.00000001
-80	0.000000001
-90	0.0000000001
-100	0.00000000001

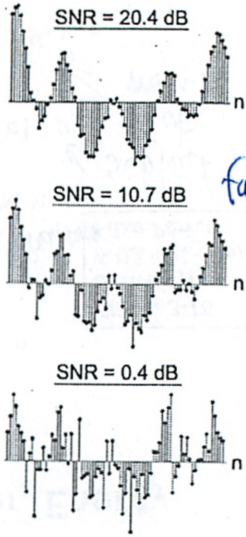
3db is a factor of 2 ✓

2/22

SNR Example



make peaks ↓ higher



factor of 10

10 x more signal than noise

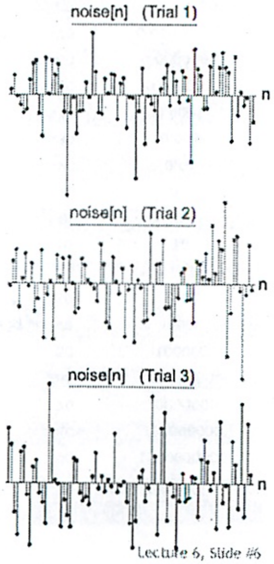
near the same

Changing the amplification factor (gain) A leads to different SNR values:

- Lower A → lower SNR
- Signal quality degrades with lower SNR

Analysis of Random Processes

- Random processes, such as noise, take on different sequences for different trials
  - Think of trials as different measurement intervals from the same experimental setup (as in lab)
- For a *given* trial, we can apply our standard analysis tools and metrics
  - mean and power calculations, etc...
- When trying to analyze the *ensemble* (i.e., all trials) of possible outcomes, we find ourselves in need of new tools and metrics



Want math model to summarize all the trials

2 Properties

Stationary and Ergodic Random Processes

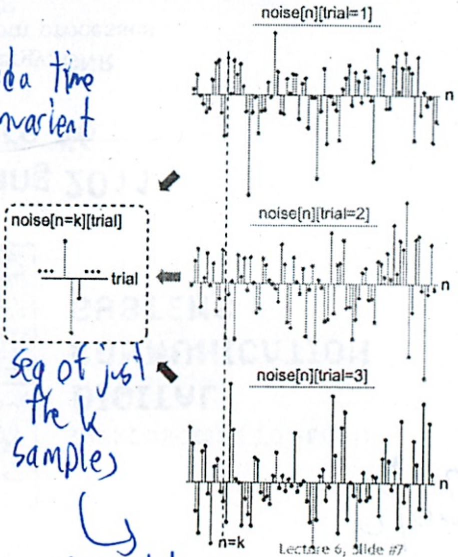
Stationary

statistical behavior is independent of shifts in time in a given trial. Implies noise[k] is statistically indistinguishable from noise[k+N]

kinda time invarient

Ergodic

statistical sampling can be performed at one sample time (i.e., n=k) across different trials, or across different sample times of the same trial with no change in the measured result

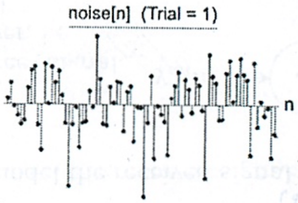


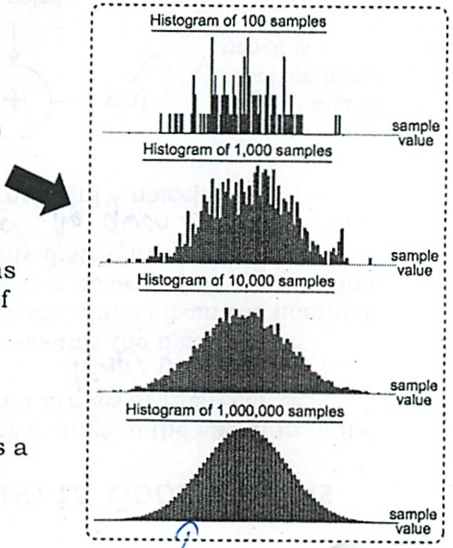
seq of just the k samples

Same stats as

doesn't change where it is in process

Experiment to See Statistical Distribution





Experiment: create histograms of sample values from trials of increasing lengths.

Assumption of stationarity implies histogram should converge to a shape known as a probability density function (PDF)

Converges to Normal

Formalizing the PDF Concept

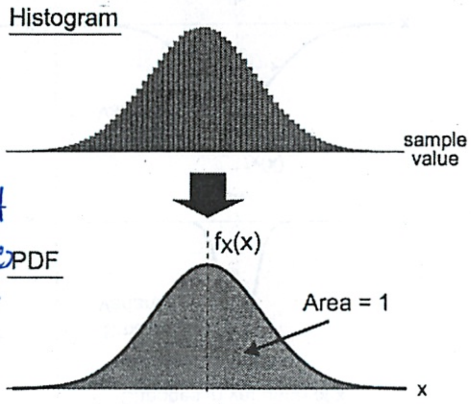
Define x as a random variable whose PDF has the same shape as the histogram we just obtained.

Denote the PDF of x as $f_x(x)$ and scale $f_x(x)$ such that its overall area is 1:

equation that describes curve

$$\int_{-\infty}^{\infty} f_x(x) = 1$$

area must = 1

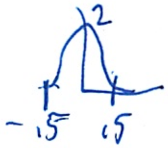


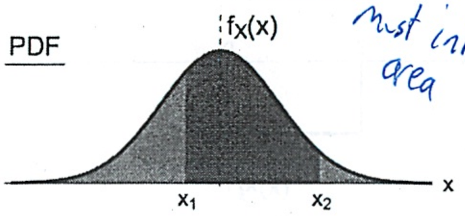
Formalizing Probability

The probability that random variable x takes on a value in the range of x_1 to x_2 is calculated from the PDF of x as:

$$p(x_1 \le x \le x_2) = \int_{x_1}^{x_2} f_x(x)dx$$

Can have

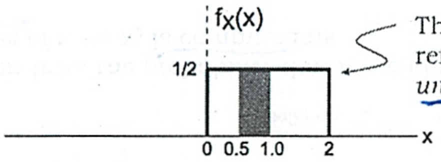




must integrate area

Note that probability values are always in the range of 0 to 1.

Example Probability Calculation



This shape is referred to as a *uniform* PDF.

Verify that overall area is 1:

$$\int_{-\infty}^{\infty} f_x(x)dx = \int_{0}^{2} 0.5\,dx = 1$$

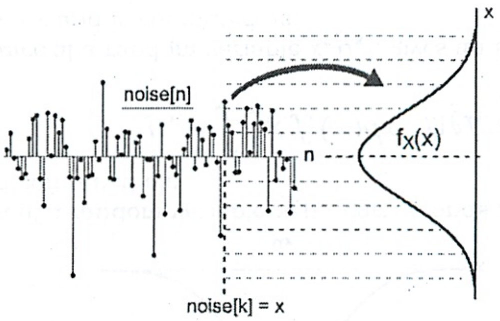
Probability that x takes on a value between 0.5 and 1:

* must integrate

$$p(0.5 \le x \le 1.0) = \int_{0.5}^{1} 0.5\,dx = 0.25$$

Can't just read it

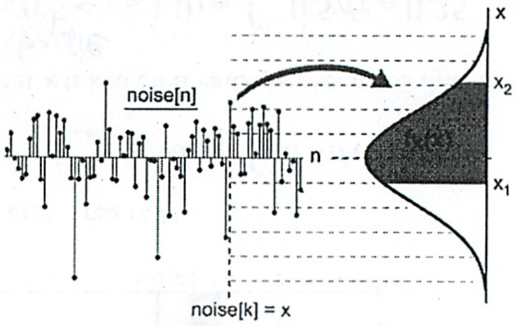
Examination of Sample Value Distribution



Assumption of ergodicity implies the value occurring at a given time sample, noise[k], across many different trials has the same PDF as estimated in our previous experiment of many time samples and one trial.

Thus we can model noise[k] using the random variable x.

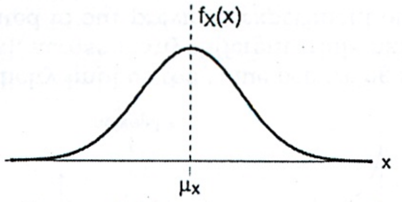
Probability Calculation



In a given trial, the probability that noise[k] takes on a value in the range of x_1 to x_2 is computed as

$$p(x_1 \le x \le x_2) = \int_{x_1}^{x_2} f_x(x)\,dx$$

Mean and Variance



The *mean* of a random variable x, μ_x, corresponds to its average value and computed as:

$$\mu_x = \int_{-\infty}^{\infty} x\, f_x(x)\,dx$$

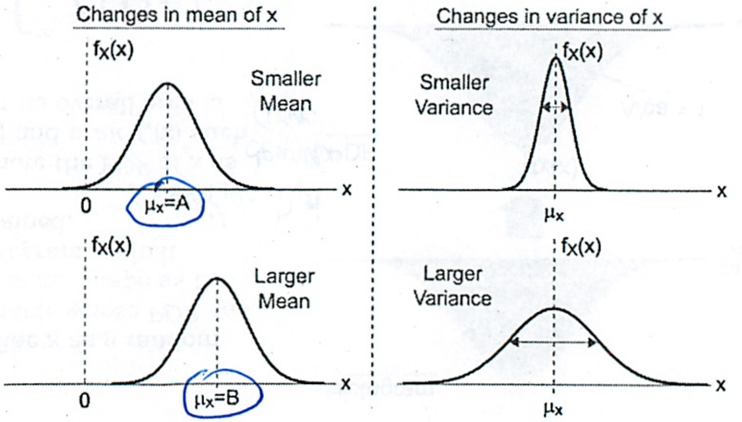
mean shall be 12 for noise

The *variance* of a random variable x, σ_x^2, gives an indication of its variability and is computed as:

$$\sigma_x^2 = \int_{-\infty}^{\infty} (x-\mu_x)^2 f_x(x)\,dx$$

Compare with power calculation

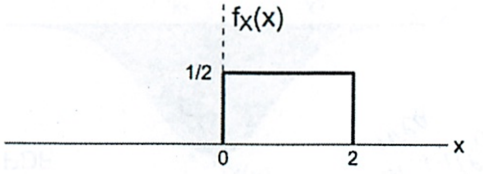
Visualizing Mean and Variance



Changes in mean shift the center of mass of PDF

Changes in variance narrow or broaden the PDF (but area is always equal to 1)

Example Mean and Variance Calculation



Mean:

$$\mu_x = \int_{-\infty}^{\infty} x\, f_x(x)\,dx = \int_0^2 x\frac{1}{2}dx = \frac{1}{4}x^2\Big|_0^2 = 1$$

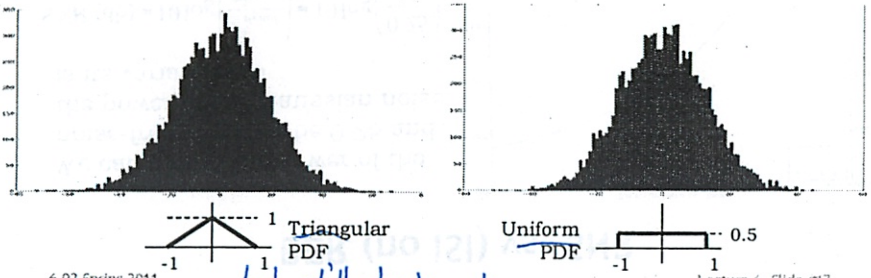
Variance:

$$\sigma_x^2 = \int_{-\infty}^{\infty} (x-\mu_x)^2 f_x(x)\,dx = \int_0^2 (x-1)^2\frac{1}{2}dx = \frac{1}{6}(x-1)^3\Big|_0^2 = \frac{1}{3}$$

Noise on a Communication Channel

The net noise observed at the receiver is often the sum of many small, independent random contributions from the electronics and transmission medium. If these independent random variables have finite mean and variance, the Central Limit Theorem says their sum will be *normally* distributed.

The figure below shows the histograms of the results of 10,000 trials of summing 100 random samples draw from [-1,1] using two different distributions.



but still tends to be normal distributed

better in color

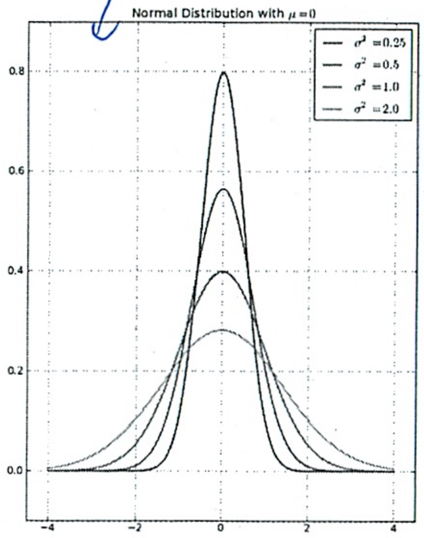
The Normal Distribution

A normal or Gaussian distribution with mean μ and variance σ² has a PDF described by

PDF normal

$$f_x(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

The normal distribution with μ=0 and σ²=1 is called the "standard" or "unit" normal.



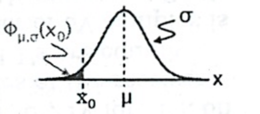
all 0 mean

Cumulative Distribution Function ϕ

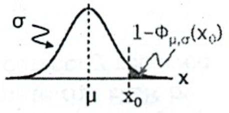
When analyzing the effects of Gaussian noise, we'll often want to determine the probability that the noise is larger or smaller than a given value x_0. From slide #10:

CDF normal

$$p(x \le x_0) = \int_{-\infty}^{x_0} \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} dx \equiv \Phi_{\mu,\sigma}(x_0)$$



$$p(x \ge x_0) = \int_{x_0}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} dx = 1 - \Phi_{\mu,\sigma}(x_0)$$



Where $\Phi_{\mu,\sigma}(x)$ is the cumulative distribution function (CDF) for the normal distribution with mean μ and variance σ². The CDF for the unit normal is usually written as just Φ(x).

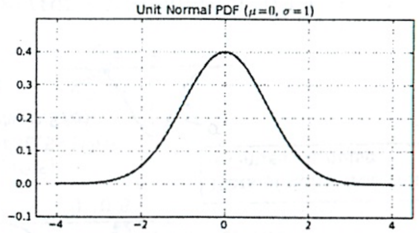
$$\boxed{\Phi_{\mu,\sigma}(x) = \Phi\left(\frac{x-\mu}{\sigma}\right)}$$

convert for unit normal

Φ(x) = CDF for Unit Normal PDF

Most math libraries don't provide Φ(x) but they do have a related function, erf(x), the *error function:*

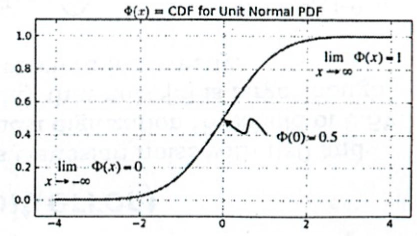
$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$



For Python hackers:

```
from math import sqrt
from scipy.special import erf

# CDF for Normal PDF
def Phi(x,mu=0,sigma=1):
  t = erf((x-mu)/(sigma*sqrt(2)))
  return 0.5 + 0.5*t
```



Bit Error Rate

fraction of # of bits in error

The *bit error rate* (BER), or perhaps more appropriately the *bit error ratio*, is the number of bits received in error divided by the total number of bits transferred. We can estimate the BER by calculating the probability that a bit will be incorrectly received due to noise.

Using our normal signaling strategy (0V for "0", 1V for "1"), on a noise-free channel with no ISI, the samples at the receiver are either 0V or 1V. Assuming that 0's and 1's are equally probable in the transmit stream, the number of 0V samples is approximately the same as the number of 1V samples. So the mean and power of the noise-free received signal are

$$\mu_{y_{nf}} = \frac{1}{N}\sum_{n=1}^{N} y_{nf}[n] = \frac{1}{N}\frac{N}{2} = \frac{1}{2}$$

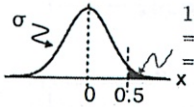
$$\tilde{P}_{y_{nf}} = \frac{1}{N}\sum_{n=1}^{N}\left(y_{nf}[n]-\frac{1}{2}\right)^2 = \frac{1}{N}\sum_{n=1}^{N}\left(\frac{1}{2}\right)^2 = \frac{1}{N}\frac{N}{4} = \frac{1}{4}$$

Use prob to predict error rate

p(bit error)

Now assume the channel has Gaussian noise with μ=0 and variance σ^2. And we'll assume a digitization threshold of 0.5V. We can calculate the probability that noise[k] is large enough that y[k] = y_{nf}[k] + noise[k] is received incorrectly:

p(error | transmitted "0"):

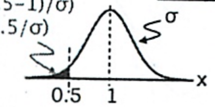


$1-\Phi_{\mu,\sigma}(0.5) = \Phi_{\mu,\sigma}(-0.5)$
$= \Phi((-0.5-0)/\sigma)$
$= \Phi(-0.5/\sigma)$

Plots of noise-free voltage + Gaussian noise

p(error | transmitted "1"):

$\Phi_{\mu,\sigma}(0.5)$
$= \Phi((0.5-1)/\sigma)$
$= \Phi(-0.5/\sigma)$



p(bit error) = p(transmit "0")*p(error | transmitted "0") +
p(transmit "1")*p(error | transmitted "1")
= 0.5*Φ(−0.5/σ) + 0.5*Φ(−0.5/σ)
= Φ(−0.5/σ)

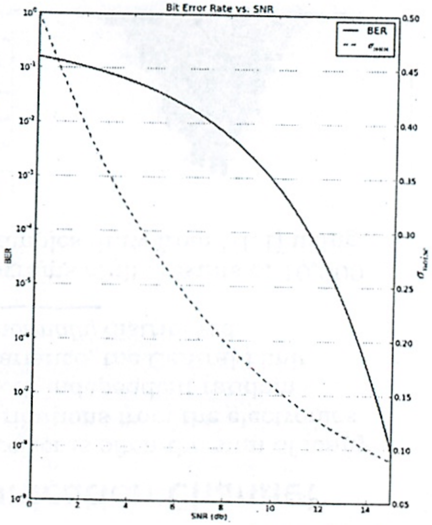
BER (no ISI) vs. SNR

We calculated the power of the noise-free signal to be 0.25 and the power of the Gaussian noise is its variance, so

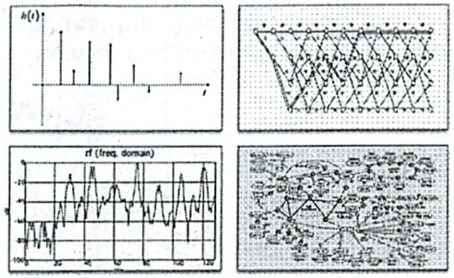
$$\text{SNR (db)} = 10\log\left(\frac{\tilde{P}_{signal}}{\tilde{P}_{noise}}\right) = 10\log\left(\frac{0.25}{\sigma^2}\right)$$

Given an SNR, we can use the formula above to compute σ^2 and then plug that into the formula on the previous slide to compute p(bit error) = BER.

The BER result is plotted to the right for various SNR values.



2/23



INTRODUCTION TO EECS II

DIGITAL COMMUNICATION SYSTEMS

6.02 Spring 2011 Lecture #7

- ISI and BER
- Choosing V_{th} to minimize BER

Bit Error Rate

The *bit error rate* (BER), or perhaps more appropriately the *bit error ratio*, is the number of bits received in error divided by the total number of bits transferred. We can estimate the BER by calculating the probability that a bit will be incorrectly received due to noise.

But more interested in samples than bits

Using our normal signaling strategy (0V for "0", 1V for "1"), on a noise-free channel with no ISI, the samples at the receiver are either 0V or 1V. Assuming that 0's and 1's are equally probable in the transmit stream, the number of 0V samples is approximately the same as the number of 1V samples. So the mean and power of the noise-free received signal are

mean $$\mu_{y_{nf}} = \frac{1}{N}\sum_{n=1}^{N} y_{nf}[n] = \frac{1}{N}\frac{N}{2} = \frac{1}{2}$$

power $$\tilde{P}_{y_{nf}} = \frac{1}{N}\sum_{n=1}^{N}\left(y_{nf}[n]-\frac{1}{2}\right)^2 = \frac{1}{N}\sum_{n=1}^{N}\left(\frac{1}{2}\right)^2 = \frac{1}{N}\frac{N}{4} = \frac{1}{4}$$

$$x[n] \to \boxed{h[n]} \to \overset{y_{nf}[n]}{\oplus} \to y[n]$$

noise[n] not exact, but can get some info → Gaussian

Var = σ^2

Have some value of interest

→ Tells us $P(\text{noise}[k] \leq x_0)$

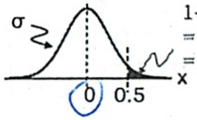
So take CDF $\Phi_{\mu,\sigma}(x) = \Phi\left(\frac{x-\mu}{\sigma}\right)$

p(bit error)

Now assume the channel has Gaussian noise with μ=0 and variance σ^2. And we'll assume a digitization threshold of 0.5V. We can calculate the probability that noise[k] is large enough that $y[k] = y_{nf}[k] + \text{noise}[k]$ is received incorrectly:

Some noise will make 0 more neg but other noise will be on other side of threshhold

p(error | transmitted "0"):



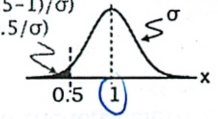
$1-\Phi_{\mu,\sigma}(0.5) = \Phi_{\mu,\sigma}(-0.5)$
$= \Phi((-0.5-0)/\sigma)$
$= \Phi(-0.5/\sigma)$

Plots of noise-free voltage + Gaussian noise

use CDF to find prob of this

p(error | transmitted "1"):

$\Phi_{\mu,\sigma}(0.5)$
$= \Phi((0.5-1)/\sigma)$
$= \Phi(-0.5/\sigma)$



p(bit error) = p(transmit "0")*p(error | transmitted "0") +
p(transmit "1")*p(error | transmitted "1")
= 0.5*Φ(−0.5/σ) + 0.5*Φ(−0.5/σ)
= Φ(−0.5/σ)

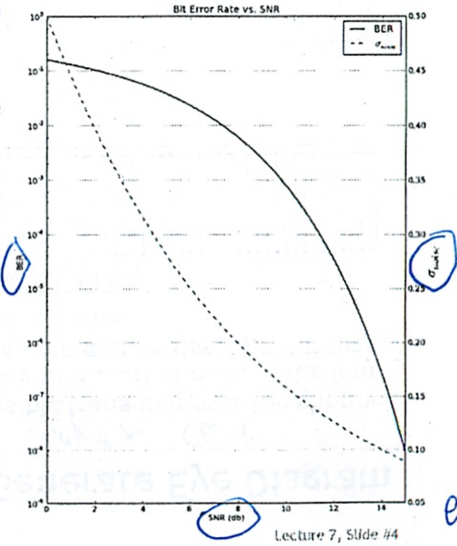
BER (no ISI) vs. SNR

We calculated the power of the noise-free signal to be 0.25 and the power of the Gaussian noise is its variance, so

$$\text{SNR (db)} = 10\log\left(\frac{\tilde{P}_{signal}}{\tilde{P}_{noise}}\right) = 10\log\left(\frac{0.25}{\sigma^2}\right)$$

Given an SNR, we can use the formula above to compute σ^2 and then plug that into the formula on the previous slide to compute p(bit error) = BER.

The BER result is plotted to the right for various SNR values.



ethernet

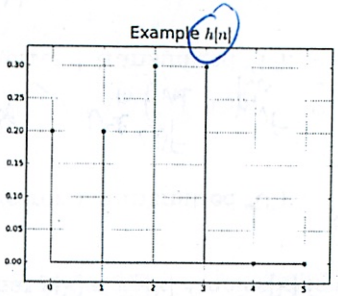
(What I had questions on)

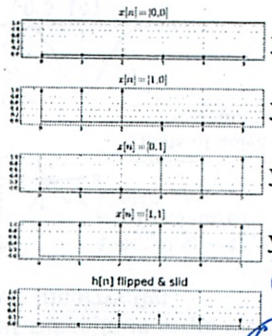
Symbols not perfect 0s or 1s

Intersymbol Interference and BER

Consider transmitting a digital signal at 3 samples/bit over a channel whose h[n] is shown on the left below.

h[] longer than Samples/bit





y[5]=0.0V

y[5]=0.3V

y[5]=0.7V

y[5]=1.0V

Each possible combo of 2 bits

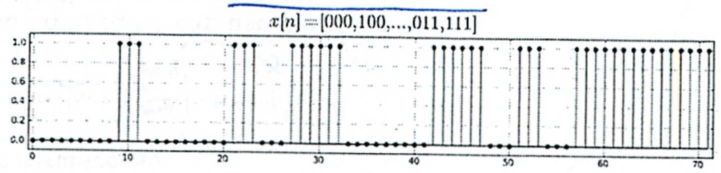
Convolution

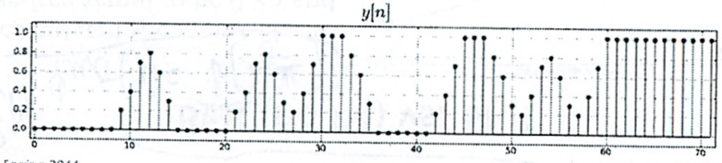
The figure on the right shows that at end of transmitting each bit, the voltage y[n] corresponding to the last sample in the bit will have one of 4 values and depends only on the current bit and previous bit.

Test Sequence to Generate Eye Diagram

So a more complex case

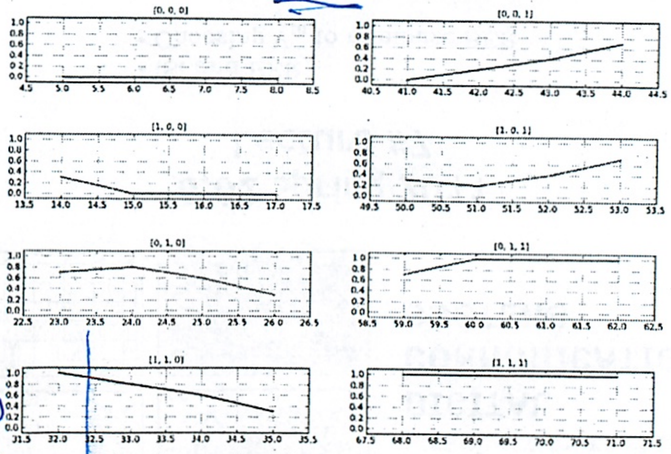
If we want to explore every possible transition over the channel, we'll need to consider transitions that start at each of the four voltages from the previous slide, followed by the transmission of a "0" and a "1", i.e., all patterns of 3 bits.





Start at above
Then add a 0 or a 1

The Eight Cases



Voltage going up even thought you are transmitting 0

The first two bits determine the starting voltage, the third bit is the test bit. The plots show the response to the test bit. All bits transmitted at 3 samples/bit.

lots of energy left from previous 1 bit

Stick on top of each other

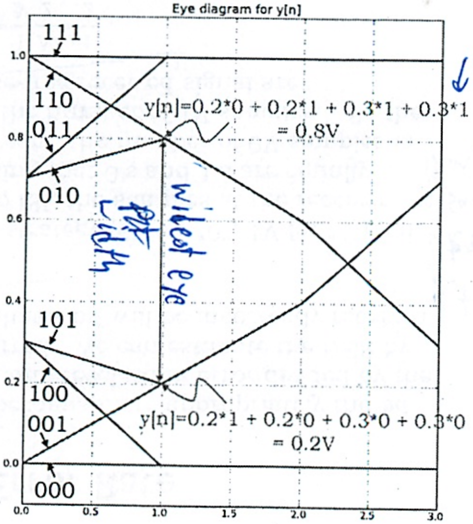
Plot the Eye Diagram

To make an eye diagram, overlay the eight plots in a single diagram.

We can label the plot with the bit sequence that generated each line.

The widest part of the eye comes at the first sample in each bit.

Using the convolution sum we can compute the width of the eye = 0.8-0.2 = 0.6V



y[n]=0.2*0 + 0.2*1 + 0.3*1 + 0.3*1 = 0.8V

y[n]=0.2*1 + 0.2*0 + 0.3*0 + 0.3*0 = 0.2V

find it w/ convol sum

What are possible voltages at widest part of eye?
What are prob for each?

BER and ISI

From the diagram on the previous slide, if we sample at the widest point in the eye, the noise-free signal will produce one of four possible samples:

1. 1.0V if last two bits are “11”
2. 0.8V if last two bits are “10”
3. 0.2V if last two bits are “01”
4. 0.0V if last two bits are “00”

Since all the sequences are equally likely, the probability of observing a particular voltage is 0.25.

Let’s repeat the calculation of p(bit error), this time on a channel with ISI, assuming Gaussian noise with a variance of σ^2 (from now on we’ll assume that Gaussian noise has a mean of 0). Again, we’ll use a digitization threshold of 0.5V.

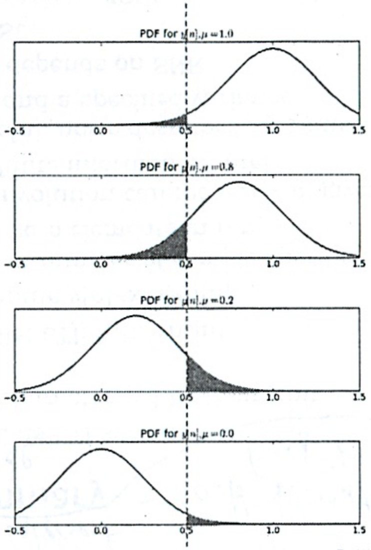
p(bit error) with ISI

p(error | 11) = $\Phi((0.5-1.0)/\sigma)$
= $\Phi(-0.5/\sigma)$

p(error | 10) = $\Phi((0.5-0.8)/\sigma)$
= $\Phi(-0.3/\sigma)$

p(error | 01) = $1-\Phi((0.5-0.2)/\sigma)$
= $\Phi(-0.3/\sigma)$

p(error | 00) = $\Phi((0.5-1)/\sigma)$
= $\Phi(-0.5/\sigma)$



p(bit error) with ISI cont’d.

$$\begin{aligned}\text{p(bit error)} &= \text{p(11)*p(error | 11)} + \text{p(10)*p(error | 10)} + \\ &\quad \text{p(01)*p(error | 01)} + \text{p(00)*p(error | 00)} \\ &= 0.25*\Phi(-0.5/\sigma) + 0.25*\Phi(-0.3/\sigma) + \\ &\quad 0.25*\Phi(-0.3/\sigma) + 0.25*\Phi(-0.5/\sigma) \\ &= 0.5*\Phi(-0.5/\sigma) + 0.5*\Phi(-0.3/\sigma)\end{aligned}$$

as eye closes - the noise makes higher prob of bit error - bits read as wrong bit

Suppose σ=0.25. Compare the formula above to the formula on slide #3 to determine what ISI has cost us in terms of BER:

p(bit error, no ISI) = $\Phi(-0.5/0.25) = \Phi(-2) = 0.023$

p(bit error, with ISI) = $0.5*\Phi(-2) + 0.5*\Phi(-1.2) = 0.069$8 7% - crummy channel

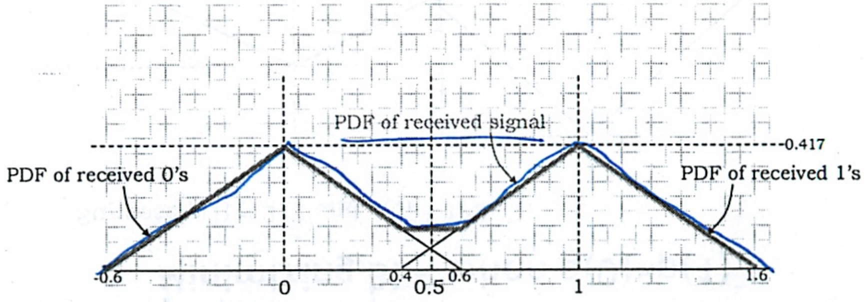
Bottom line: a factor of 3 increase in BER 3x worse

Want Threshhold to minimize error rate

Choosing V_{th}

We’ve been using 0.5V as the digitization threshold – it’s the voltage half-way between the two signaling voltages of 0V and 1V. Assuming that the probability of transmitting 0’s and 1’s is the same, this choice minimizes the BER. Let’s see why…

Suppose noise has a triangular distribution from -0.6V to 0.6V:

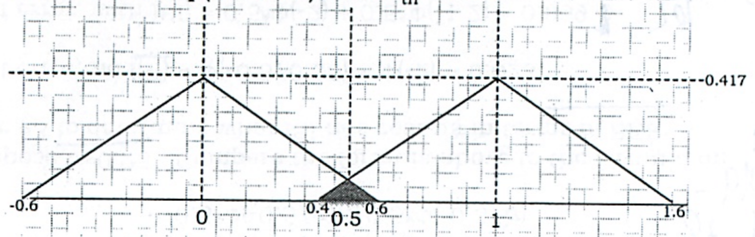


Aggressivly moving through the formulas

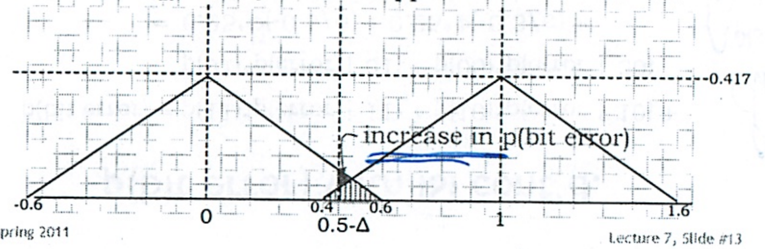
Equal Prob

Minimizing BER

Shaded area = p(bit error) with V_{th} = 0.5V



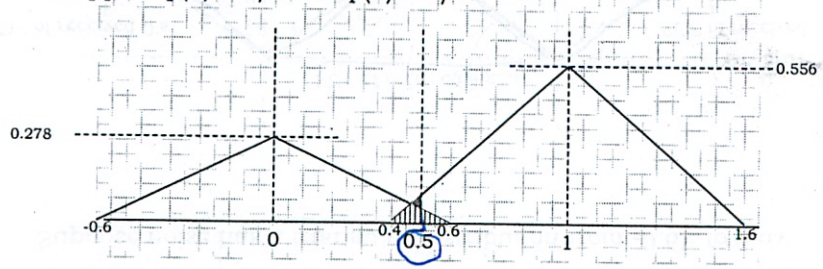
Now move V_{th} slightly. What happens to BER?



Not = Prob

Minimizing BER when p(0)≠p(1)

Suppose p(1) = 2/3 and p(0) = 1/3:



If we leave V_{th} at 0.5V, we can see that p(bit error) will be larger than if we moved the threshold to a lower voltage. p(bit error) will be minimized when threshold is set at intersection of the two PDFs.

Question: with triangular noise PDF, can you devise a signaling protocol that has p(bit error) = 0?

if send more 1s, build a reciever that is better at recieving 1s

put lines where P lines intersect

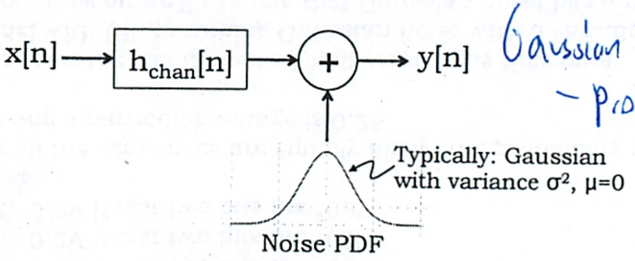
could do math to find it

→ Study Guide

Channel Model Summary

Could move this triangle to 2 and have 0 error

Gaussian is never cut off - Prob just gets smaller



The Good News: Using this model we can predict ISI and compute the BER given the SNR or σ. Often referred to as the AWGN (additive white Gaussian noise) model.

freq dist of energy

The Bad News: Unbounded noise means BER ≠ 0, i.e., we'll have bit errors in our received message. How do we fix this? Our next topic!

Will run experiments in lab

experiments will slightly match models

Summary

- Noise-free channels modeled as LTI systems
- LTI systems are completely characterized by their unit sample response h[n]
- Series LTI: $h_1[n]*h_2[n]$, parallel LTI: $h_1[n]+h_2[n]$
- Use convolution sum to compute $y[n]=x[n]*h[n]$
- Intersymbol interference when number of samples per bit is smaller than number of non-zero elements in h[n]
- In a noise-free context, deconvolution can recover x[n] given y[n] and h[n]. Potentially infinite information rate!
- With noise $y[n] = y_{nf}[n]+noise[n]$, noise described by Gaussian distribution with zero mean and a specified variance
- Bit Error Rate = p(bit error), depends on SNR
- BER = $\Phi(-0.5/\sigma)$ when no ISI
- BER increases quick with increasing ISI (narrower eye)
- Choose V_{th} to minimize BER

Past 2 lecture fairly straightforward

6.02 Recitation

2/24

Probability

$\Omega = \{0,1\}$ $\in$ universe

$\Omega = \mathbb{R} = (-\infty,\infty)$

$\mathcal{E} = \{\geq 7\}$ event

Takes all subsets of the universe

For each subset, probability assigns it a value

$\mathbb{P} : \mathcal{E}(\subseteq \Omega) \rightarrow P(\mathcal{E}) \in [0,1]$

~~$\tilde{\mathcal{E}} = \{\geq 9\} \rightarrow .04$~~ Bad example

Rules

1. $P(\mathcal{E}) \geq 0, \leq 1$
2. $P(\Omega) = 1$
3. $\mathcal{E}_1, \mathcal{E}_2$ $\mathcal{E}_1 \cap \mathcal{E}_2 = \emptyset$

$$P(\mathcal{E}_1 \cup \mathcal{E}_2) = P(\mathcal{E}_1) + P(\mathcal{E}_2)$$

② Gaussian/Normal

$X \sim N(\mu, \sigma^2)$ $\quad \mu \in \mathbb{R} \quad \sigma^2 \geq 0$

$$P(X \leq a) = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right) dx$$

$$= \Phi(a)$$

$P(Y + \mu \leq a)$

$\quad \shortparallel$

$P(Y \leq a - \mu)$ $\qquad X = \underset{N(0,\sigma^2)}{\underset{\wr}{Y}} + \mu$ $\qquad \Phi(b) = P(Y \leq b)$

$Y = \sigma Z$

$Z \sim N(0,1)$

$X \sim \underset{f}{\text{PDF}}$ $\qquad P(X = x) = f(x)dx$

$\qquad P(X \leq x) = \int_{-\infty}^{x} f(y)\,dy$

$$E[X] = \int_{-\infty}^{\infty} x f(x) dx$$

$$\text{var}(X) = E[(X - E(X))^2]$$

③

$Y = X + \mu$

$E[Y] = E[X] + \mu$

$Y = \sigma Z$

$\text{var}(X) = \sigma^2 \text{var}(Z)$

$X = \sigma \underset{\substack{\uparrow \\ N(0,1)}}{Z} + \mu$

Take away message

$X \sim N(\mu, \sigma^2)$

$X = \sigma Z + \mu$

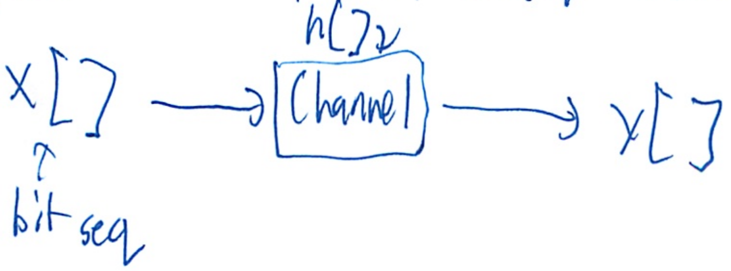
$Z \sim N(0,1)$

$$P(X \le a) = P(\sigma Z + \mu \le a) = P\left(Z \le \left(\frac{a-\mu}{\sigma}\right)\right) = \Phi\left(\frac{a-\mu}{\sigma}\right)$$

$$= \int_{-\infty}^{\frac{a-\mu}{\sigma}} \frac{1}{\sqrt{2\pi}} \cdot \exp\left(\frac{-\theta^2}{2}\right) d\theta$$

Bit error rate

P(noise so much, can't estimate correctly)



4

But what if noise is added?

$$\tilde{y}[\,] = y[\,] + n[\,]$$

Deconvolling is screwed up by noise

Get $\tilde{x}[\,]$

$$\tilde{X} - x = \tilde{n}$$

Deconvolution in a linear operation

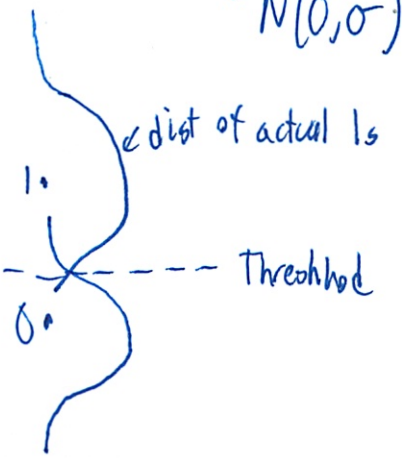
Think of noise as ~~[illegible]~~ guassian dist

Use that to do deconvolve

<u>Tutorial Poblem</u>

Suppose noise added to 0 or 1

$\uparrow N(0,\sigma)$

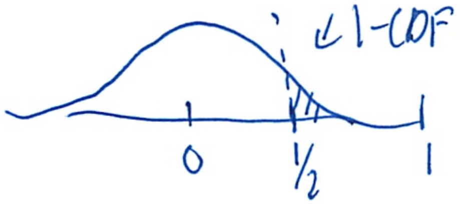


Recieve = In + Noise

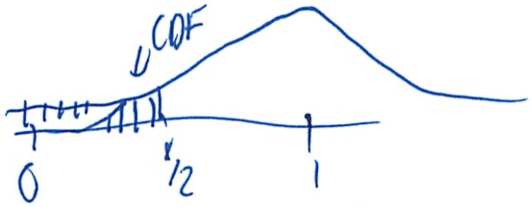
⑤

What are the chances stuff screws up

$P_{0\to1}$ 0 → 1
sent recieved



$P_{1\to0}$ 1 → 0
sent recieved



$$BER = P(0)\cdot P_{0\to1} + P(1)\cdot P_{1\to0}$$

$$.5\cdot P_{0\to1} + .5\cdot P_{1\to0}$$

$$= P_{0\to1} = P_{1\to0}$$ same, symetric

$$P_{0\to1} = P(Rec > V/2 \mid In = 0)$$

$$= P(Noise > V/2)$$

$$= P(\sigma_{Noise}\, Z > V/2)$$

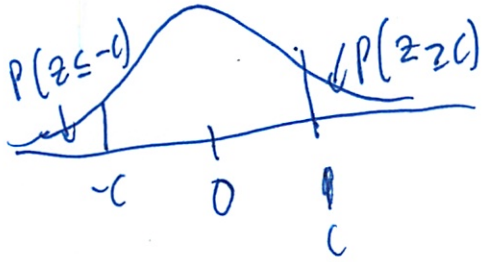
$$= P\left(Z > \frac{V}{2\sigma_{Noise}}\right)$$

$$Noise \sim N(0, \sigma^2_{Noise})$$

$$Noise = \sigma_{Noise}\, \underset{\sim N(0,1)}{Z} + 0$$

6

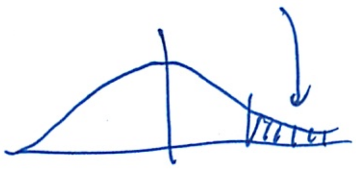
$z \sim N(0,1)$



$P(z \ge c) = P(z \le c)$

Then what is $-z$? Same

So do $1-\phi(c)$



$$P\left(-z \le \frac{V}{2\sigma_{Noise}}\right) = \phi\left(\frac{V}{-2\sigma_{Noise}}\right) = 1-\phi\left(\frac{V}{2\sigma_{Noise}}\right)$$

$$P_{1\to0} = P\left(Rec < \frac{V}{2} \,\middle|\, In = V\right)$$

$$= P\left(Noise < -\frac{V}{2}\right)$$

$$= P\left(\sigma_{Noise} z < -\frac{V}{2}\right)$$

$$= P\left(z \le \frac{V}{-2\sigma_{Noise}}\right)$$

$$= 1-\phi\left(\frac{V}{2\sigma_{Noise}}\right)$$

So $BER = 1-\phi\left(\frac{V}{2\sigma_{Noise}}\right)$

7

(I get all the concepts from lecture, but the notation he uses is weird)

When = # 0 or 1 put threshold in middle

When V ↑, this quantity ↑

↳ $1 - \phi\left(\frac{V}{2\sigma_{Noise}}\right)$

V is the voltage 1 is sent at

↳ the difference from 0 which is sent at 0

To calculate σ

- send test signals
- try to measure how much signal has changed

Its relative SNR ratio that matters

If >0, can transmit something

But not very efficiently

> To save your work, click the SAVE button at the bottom of this page. You can revisit this page, revise your answers and SAVE as often as you like.
>
> To submit the assignment, click the SUBMIT button at the bottom of this page. YOU CAN SUBMIT ONLY ONCE. Once the assignment has been submitted, you can continue to view this page but will no longer be able to make any changes to your answers.

6.02 Spring 2011: Plasmeier,Michael E.

PSet PS3

Dates & Deadlines

issued:	Feb-16-2011 at 00:00
due:	Feb-24-2011 at 06:00 (Mar-01-2011 at 06:00 with extension)
checkoff due:	Mar-01-2011 at 06:00

Help is available from the staff in the 6.02 lab (38-530) during lab hours -- for the staffing schedule please see the Lab Hours page on the course website. We recommend coming to the lab if you want help debugging your code.

For other questions, please try the 6.02 on-line Q&A forum at Piazzza.

Your answers will be graded by actual human beings, so your answers aren't limited to machine-gradable responses. Some of the questions ask for explanations and it's always good to provide a short explanation of your answer.

Each question in this problem set deals with a causal linear time-invariant (LTI) system. We'll be working with discrete time samples and the index for the sample sequences is always an integer. For each question, assume that:

- The sequence `x[n]` is the input to a causal LTI system and `x[n] = 0` for `n < 0`.
- The sequence `h[n]` is the unit-sample response of the causal LTI system.
- The sequence `y[n]` is the output of the causal LTI system described by `h[n]`, with `x[n]` as the input.
- `δ[n]` is the unit-sample sequence: `δ[n] = 1` when `n = 0`, and zero otherwise. One
- `u[n]` is the unit-step sequence: `u[n] = 1` when `n ≥ 0`, and zero otherwise. goes up

You may find it helpful in solving the problems to sketch out the sequences described mathematically below.

σ = unit Sample

U = unit step

$\boxed{x \to \boxed{h} \to y}$

~~δ = Unit Sample~~ u = Unit step

Problem 1. (2 points)

↓what does n mean here?

Suppose `h[n] = δ[n-N]`, i.e., `h[n] = 1` for `n = N`, and zero otherwise. How will an eye diagram for this system change with increasing N? delay

TA Kirchner Confused

N=0 h[0]=δ[0-0] no delay

N=1 h[1]=δ[1-1]

just delay

(points: 1) how to do eye diagram

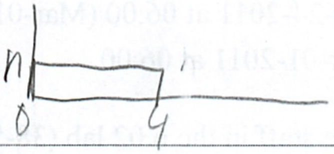
Problem 2. (1.5 points)

Suppose

unit step

`x[n] = u[n]` and

`h[n] = n` for `0 ≤ n ≤ 4`, and zero otherwise



so is that last 4 steps?

Determine `y[2]`, `y[3]`, and `y[20]`.

Student helped see paper

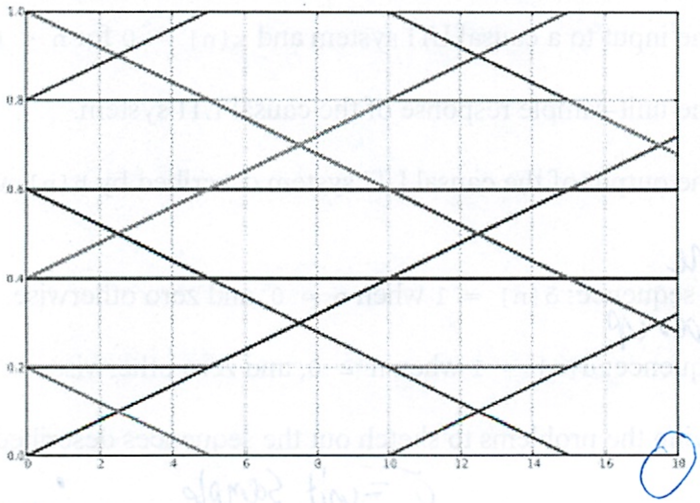
[3, 6, 10]

(points: 1.5)

Problem 3. (1.5 points)

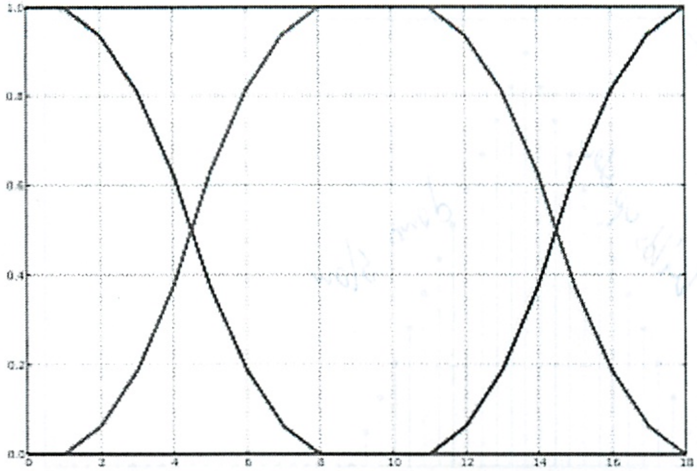
Consider the following three eye diagrams generated by applying a random sequence of 200 bits, with 10 samples/bit, to three different causal LTI systems:

Eye Diagram A



3

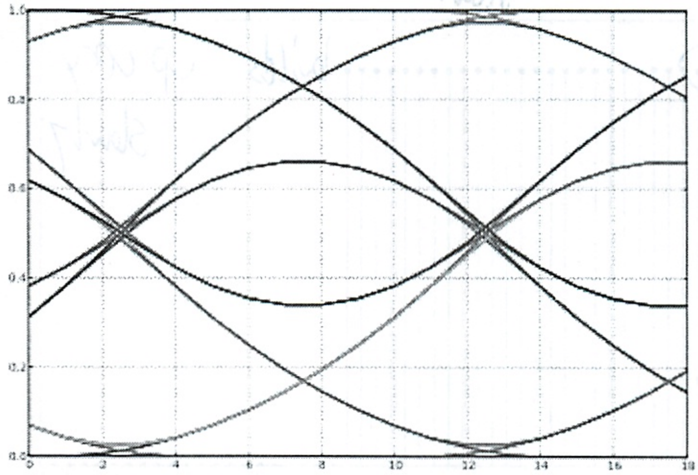
Eye Diagram B



#1

↕ if flip
-fast channel
large eye?

Eye Diagram C

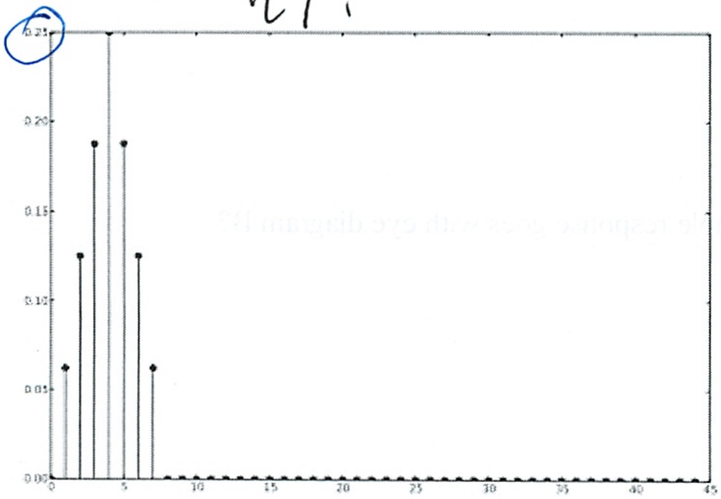


#2

The unit sample responses for each of the three causal LTI systems are given below, in some order:

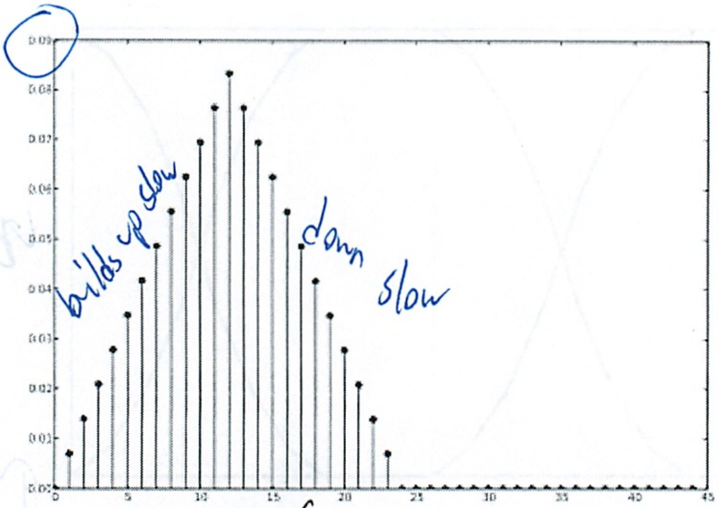
↳ is that h[7]?

Unit-sample Response 1



fast?

Unit-sample Response 2

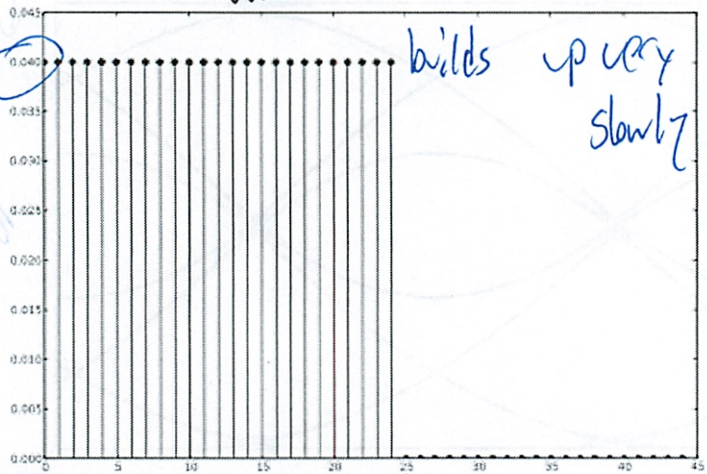


slow!

key!

note! →

Unit-sample Response 3



Put in

this is what happens

delay!

need examples!

A. Which unit sample response goes with eye diagram A?

3

(points: 0.5)

B. Which unit sample response goes with eye diagram B?

1

(points: 0.5)

C. Which unit-sample response goes with eye diagram C?

WP: eyepattern

now try ! again

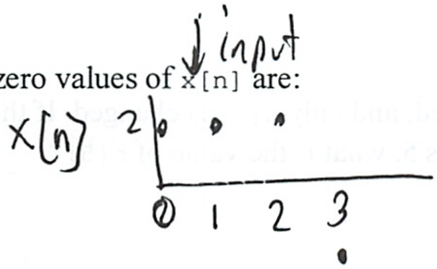
2

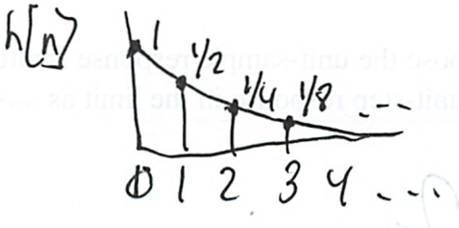
(points: 0.5)

Problem 4. (1 point)

Suppose the only nonzero values of x[n] are:

```
x[0] = 2
x[1] = 2
x[2] = 2
x[3] = -2
```





If h[n] = $(1/2)^n$ for n ≥ 0, what is the maximum value of y[n] and for what value of n does y[n] achieve its maximum?

So do convolution - see page

$Y[2] = 2 \cdot \frac{1}{4} + 2 \cdot \frac{1}{2} + 2 \cdot 1 = 3.5$

(points: 1)

Problem 5. (1 point)

Suppose

```
x[n] = (1/2)^n for n ≥ 0,
y[0] = 4, and
y[1] = 1
```

If the LTI system is causal, what are the values of h[0] and h[1]?

4
-1

(points: 1)

Problem 6. (2 points) Suppose the only nonzero values of a unit-sample response are

```
h[0] = 1
h[1] = 2
h[2] = -1
h[3] = 1/2
```

```
h[4] = 1/2
h[n] = 0 for n > 4
```

A. What are the values of the unit-step response s[n]?

See paper

(points: 1)

B. Suppose the unit-sample response is altered, and only h[5] is changed. If the value of the unit-step response in the limit as n→∞ is 5, what is the value of h[5]?

2

(points: 1)

Problem 7. (1 point)

Suppose the unit-step response, s[n], is given by

```
s[0] = 0
s[1] = 0.1
s[2] = 0.5
s[3] = 0.9
s[n] = 1.0 for n &g3; 4
```

7

? is this the response to u[n]

Yes

Determine h[n] for 0 ≤ n ≤ 10.

See sheet

(points: 1)

Problem 8. (3 points)

Suppose a linear time-invariante channel has a unit sample response:

```
h[n] = 0.5    n = 0, 1, 2
h[n] = 0      otherwise
```

If the output of the channel is

```
y[n] = 1      n = 0, 1
```

```
y[n] = 0      otherwise
```

getting repetitive

Please determine the value of the first three voltage samples of the input to the channel: `x[0]`, `x[1]`, and `x[2]`.

See sheet

(points: 3)

Python Task 1: Unit Sample Response of a Channel (1 point)

Useful download links:

PS3_tests.py -- test jigs for this assignment
PS3_1.py -- template file for this task

In lecture we saw that the unit sample response completely characterizes the effect of a channel on sequences of samples that pass through the channel. So, determining the unit sample response of a channel is a handy way to model a channel. The unit sample response is also useful in figuring out how to engineer the receiver to compensate for a channel's less desirable effects.

There is a small complication: formally, the unit sample response (USR) extends for an infinite number of samples. For all practical purposes, the USR will be so close to zero after a modest number of samples, that a USR-based model of a channel will still be quite accurate even if we truncate the response to a finite number of samples (as long as the samples beyond the truncation point are sufficiently close to zero). To find a reasonable point to truncate a USR, note that you'll need to start with a USR that is much longer than the truncated USR.

For this task, write a Python function `unit_sample_response` that returns a truncated sequence of samples that corresponds to the unit sample response of the specified channel:

truncate the useless end part

```
response = unit_sample_response(mychannel,max_length=1000,tol=0.005)
```

The *`mychannel`* argument will be a channel instance which you can call like a function, passing in a sequence of voltage samples representing the input sequence. It will return a sequence of voltage samples representing the channel's response to that input.

`max_length=1000` sets an upper limit on the number of samples to be used to represent a unit sample response.

TA: Baddly written!

tallest?

`tol=0.005` sets the accuracy criterion used for truncating the unit sample response. To perform the truncation, we first need determine the largest *magnitude* sample in the response, call it `h_max`. If `tol > 0`, the response sequence should be truncated to `h[0:K]` (using Python slice notation) if the magnitude of every sample in `h[K:]` (the part of `h` we're eliminating) is smaller than `tol*h_max` and the magnitude of `h[K-1]` is

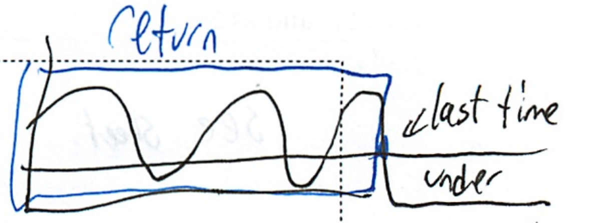
after that

what is k?

≥ `tol*h_hax`. If `tol = 0`, the response sequence should be of length `max_length`.

PS3_1.py is a template file for this task:

TA badly worded



```
# template for PSet #3, Python Task #1
import numpy
import matplotlib.pyplot as p
import PS3_tests

# arguments:
#   channel -- instance of the PS3_tests.channel class
#   max_length -- integer
#   tol -- float
# return value:
#   a voltage sequence of length max_length or less
def unit_sample_response(channel,max_length=1000,tol=0.005):
    """
    Returns sequence of samples that corresponds to the unit-sample
    response (USR) of the channel.

    channel is function you call with an input sequence of voltage
    samples.  It returns a sequence of voltage values, which is the
    response of the channel to that input.

    max_length sets the length of the test waveform to be sent through
    the channel.

    The voltage sample sequence representing the unit sample response
    should truncated to the smallest length such that the maximum
    magnitude of the truncated samples are smaller than tol times the
    value that is the largest magnitude sample in the unit sample
    response. Please make sure that if tol=0, the return USR should
    have length equal to max_length.
    """
    pass # Your code here.

if __name__ == '__main__':
    # Create the channels (noise free)
    channel0 = PS3_tests.channel(channelid='0')
    channel1 = PS3_tests.channel(channelid='1')
    channel2 = PS3_tests.channel(channelid='2')

    # plot the unit-sample response of our three virtual channels
    PS3_tests.plot_USR(unit_sample_response(channel0),'0')
    PS3_tests.plot_USR(unit_sample_response(channel1),'1')
    PS3_tests.plot_USR(unit_sample_response(channel2),'2')

    p.show()
```

read - need to pass in input

what is this?

Once it dips below this

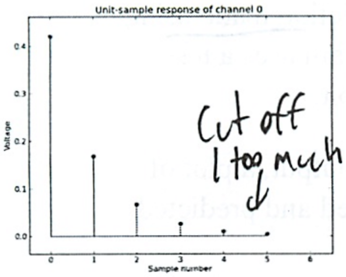
Worked but very slow

Calling input each time - save as a variable

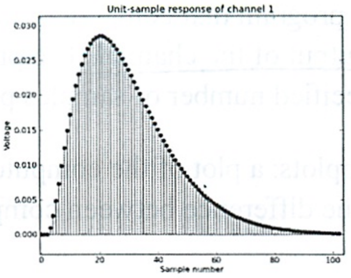
The test code uses your function to compute the unit sample response of our three channels and plots the result. If your function is working correctly you should see something like

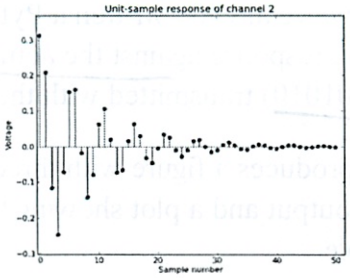
interesting how slow that went

and how much faster I made it



Cut off 1 too much





Please save and upload the three plots of the unit sample responses. To save a plot, click on the floppy disk icon in the plot window and in the dialog box that pops up, enter a name to use for the saved image.

Upload figure for USR of channel 0: ✓ Browse...

(points: 0.33)

Upload figure for USR of channel 1: ✓ Browse...

(points: 0.33)

Upload figure for USR of channel 2: Browse...

(points: 0.34) ✓

messes me up!

Communication engineers would call Channel 0 a "fast" channel, Channel 1 a "slow" channel and Channel 2 a "ringing" channel, referring to the shape of the channel's unit step response. Looking at the unit sample responses that you graphed, briefly describe what it is about the response that causes the channel to be fast, slow or ringing.

fairly obvious---

(points: 1)

Python Task 2: Predicting channel response using h[n] (1 point)

Useful download link:

PS3_2.py -- template file for this task

We also saw in lecture that given the unit sample response we can compute the response of the channel for an arbitrary input by performing the appropriate convolution sum.

For this task, we've already written a Python program that compares a prediction made using the unit sample response against the actual output of the channel. The program uses a test message (10101010) transmitted with the specified number of samples per bit.

The program produces a figure with three subplots: a plot of the computed output, a plot of the predicted output and a plot showing the the difference between computed and predicted for each sample.

PS3_2.py is the file for this task:

```
# template for PSet #3, Python Task #2
import numpy
import matplotlib.pyplot as p
import PS3_tests
from PS3_1 import unit_sample_response

# create some plots showing how prediction of the
# response using the convolution sum compares with
# the actual response from the channel.
def compare_usr_chan(channel,samples_per_bit):
    # Get channel's unit sample response
    h = unit_sample_response(channel)

    # send test message through channel
    bits = [1,0,1,0,1,0,1,0]
    test_samples = PS3_tests.transmit(bits,samples_per_bit)
    out_samples = channel(test_samples)
```

✓ don't have to build convolve

what is channel doing that conv isn't

```
    # make prediction of result using unit-sample response
    out_conv_samples = numpy.convolve(numpy.array(test_samples),
                                      numpy.array(h))

    # only compare as many samples as in the shortest output
    num_compare = min(len(out_samples),len(out_conv_samples))
    out_conv_samples = out_conv_samples[:num_compare]
    out_samples = out_samples[:num_compare]

    # plot the results
    max_ot = max([max(test_samples),
                  max(out_samples),
                  max(out_conv_samples)])
    min_ot = min([min(test_samples),
                  min(out_samples),
                  min(out_conv_samples)])
    delta = max_ot - min_ot
    plot_max = max_ot + 0.1 * delta  # avoid samples at the edges of plot
    plot_min = min_ot - 0.1 * delta

    p.figure()
    p.subplots_adjust(hspace = 0.6)
    cname = "Channel " + channel.id

    p.subplot(311)
```

```
        p.plot(out_samples)
        p.title(cname+" Output")
        p.axis([0,num_compare,plot_min,plot_max])

        p.subplot(312)
        p.plot(out_conv_samples)
        p.title(cname+" Prediction")
        p.axis([0,num_compare,plot_min,plot_max])

        p.subplot(313)
        p.plot(out_conv_samples - out_samples)
        p.title(cname+" Error")

    if __name__ == '__main__':
        # plot the unit-sample response of our two channels
        compare_usr_chan(PS3_tests.channel('1'),100)
        compare_usr_chan(PS3_tests.channel('2'),50)

        # interact with plots before exiting.
        p.show()
```

What is the prediction?

Looking at the Error plots for each channel, we see that the error is zero for a while, but at some point becomes non-zero, although not very large (look at the vertical scale for the Error plot). ← Key We wouldn't expect any error at all if the convolution sum produced a perfect prediction. Explain where the error comes from and why it's zero for a while. What's "magic" about the sample at which it becomes non-zero?

-not noise
-not sure
-Lecture?

? The 0's in the front
-right at that bump
-nope - but when it starts decreasing

- what are we doing exactly?
-the decimal points get so small??
Convolution

(points: 1)

- 1.0 exactly?

Python Task 3: Deconvolver (8 points)

Useful download link:

PS3_3.py -- template file for this task

In lecture we talked about deconvolution, an approach to reconstructing the signal at the input to the transmission system by looking at the transmission channel's output, `y[n]`, and using the channel's unit sample response, `h[n]`.

Solve for x

In particular, we showed that the sequence `w[n]` would be a reconstruction of the input sample sequence `x[n]` if `w[n]` satisfied the difference equation

Solve for, like I did earlier

```
y[n] = h[0]w[n] + h[1]w[n-1] + ... + h[K]w[n-K].
```

where `y[n]` is the sequence of channel output samples and the unit sample response `h[n]` is zero after some number of samples, i.e., `h[n] = 0` when `n > K`. 0 else

We can rearrange this equation to solve for `w[n]`: diff eq

the reconstructed input

```
w[n] = (1/h[0])(y[n] - h[1]w[n-1] - ... - h[K]w[n-K]).
```

Since you are given `y[n]`, and since you know `w[n] = 0` for `n < 0`, you can solve the above equation for `w[0]` given `y[0]`, then for `w[1]` given `w[0]` and `y[1]`, and so on: like I did above

```
w[0] = (1/h[0])(y[0])
w[1] = (1/h[0])(y[1] - h[1]w[0])
w[2] = (1/h[0])(y[2] - h[1]w[1] - h[2]w[0])
...
```

Remember to modify this simple "plug and chug" strategy when the leading values of `h[n]` (`h[0]`, `h[1]`,..) are zero. In fact the unit sample response for one of the channels has leading zeros. chop them off like lecture notes

PS3_3.py is a template for this task:

```
# template for PSet #3, Python Task #3
import numpy,random
import matplotlib.pyplot as p
import PS3_tests
from PS3_1 import unit_sample_response

# arguments:
#   y -- sequence of received voltage samples
#   h -- sequence returned by unit_sample_response
# return value
#   sequence of deconvolved voltage samples
def deconvolve(y,h):
    """
    Take the samples that are the output from a channel (y), and
    the channel's unit-sample response (h), deconvolve the
    samples, and return the reconstructed samples.  Be sure the
    length of the reconstructed samples is the same as the length
    of the input samples.

    Your code should handle the case where some number of
    the leading elements of h are zero.
    """
    pass  # your code here

if __name__ == '__main__':
    # Create two noise free channels
    mychannel1 = PS3_tests.channel(channelid='1',noise=0.0)
    mychannel2 = PS3_tests.channel(channelid='2',noise=0.0)

    # Compute the channel unit sample responses of the
    # noise-free channels
    h1 = unit_sample_response(mychannel1)
```

chop

deconvolution failed (ie, where the error grew very large as the deconvolution progressed).

Channel 1: largest noise value where deconvolution succeeded:

Browse...

(points: 0.5)

Channel 1: smallest noise value where deconvolution failed: got like 1.0e-323

Browse...

(points: 0.5)

Channel 2: largest noise value where deconvolution succeeded:

Browse...

(points: 0.5)

Channel 2: smallest noise value where deconvolution failed:

Browse...

(points: 0.5)

When you're ready, please submit the file with your code using the field below.

File to upload for Task 3: Browse...

(points: 4)

In your noise experiments, you should see that deconvolution of Channel 2 is much less sensitive to noise than deconvolution of Channel 1. Briefly explain why.

stability?

(points: 1)

You can save your work at any time by clicking the Save button below. You can revisit this page, revise your answers and SAVE as often as you like.

```
    h2 = unit_sample_response(mychannel2)

    # Generate a sequence of test samples
    samples = numpy.sin(2*numpy.pi*0.01*numpy.array(range(100)))
    samples[0:len(samples)/2] += 1.0
    samples[len(samples)/2:] += -1.0
    maxs = max(samples)
    mins = min(samples)
    samples -= mins  # Make samples positive
    samples *= 1.0/(maxs - mins) # Scale between zero and one

    # Test deconvolver
    PS3_tests.demo_deconvolve(samples,mychannel1,h1,deconvolve)
    PS3_tests.demo_deconvolve(samples,mychannel2,h2,deconvolve)

    # interact with plots before exiting
    p.show()
```

There are three parts to this deconvolution task: shald it have added 19 windows?

1. Please fill in the function `deconvolve`. The function should take a set of channel output samples and a unit sample response (from Task #1) and return the reconstructed input.

2. Run `PS3_3.py` unchanged to demonstrate that your deconvolver is working on a short sequence of test samples for two noise free channels. Please save and upload the two plots it produces: only 2

   Upload figure for demo of channel 1: Browse... ✓ now only 2

   (points: 0.5)

   Upload figure for demo of channel 2: Browse... ✓

   (points: 0.5)

3. Deconvolution is very effective if there is no noise in the transmission system. To demonstrate the impact of noise, you can add noise to the channel by changing the value of `noise` when each channel is instantiated. For example, setting `noise=1.0e-5` will add noise with an amplitude of about ±1.0e-5 to the output of the `channel`. Experiment with the noise amplitude (try values at different orders of magnitude varying from 1e-10 to 1e-1) to determine for each channel the order of magnitude for the largest noise value for which the deconvolution still succeeds. Please save and upload two plots for each channel: the plot for the largest noise value where the deconvolution still succeeded, and the plot for smallest noise value where the

   actually went well

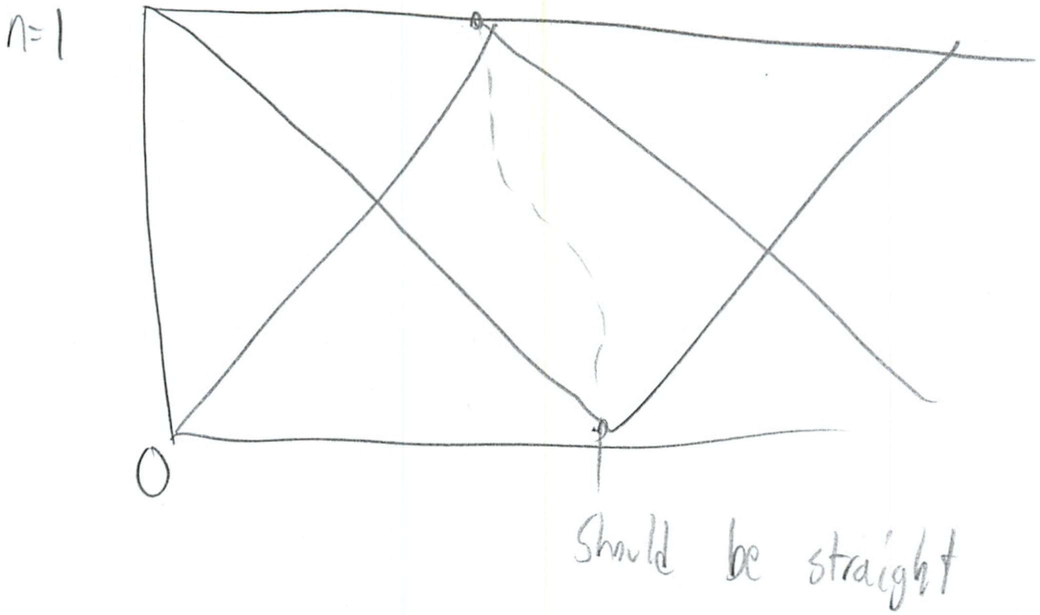
Save

To submit the assignment, click on the Submit button below. YOU CAN SUBMIT ONLY ONCE after which you will not be able to make any further changes to your answers. Once an assignment is submitted, solutions will be visible after the due date and the graders will have access to your answers. When the grading is complete, points and grader comments will be shown on this page.

Submit

2/14

$\left\{ \text{lists} \right\}$ # samples
1

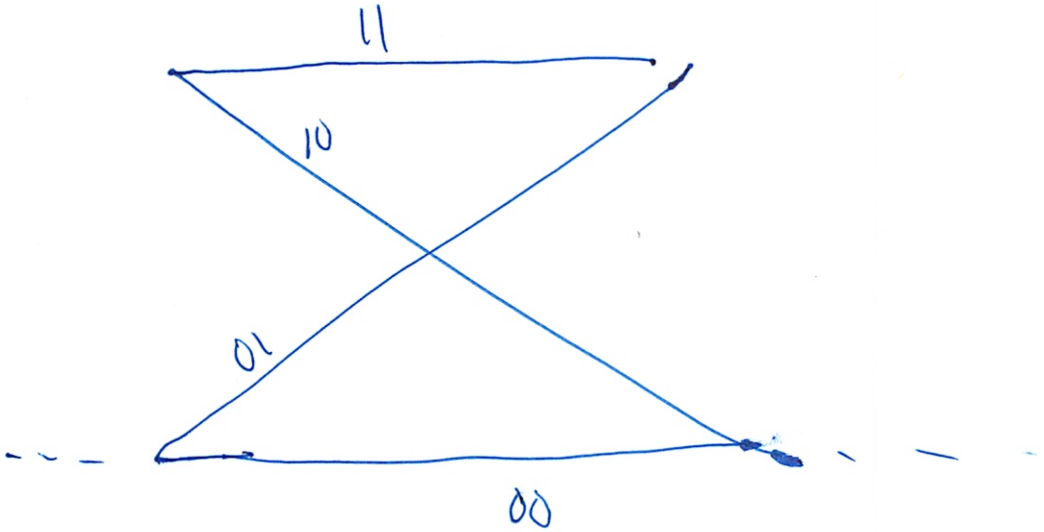


but how does delay effect?

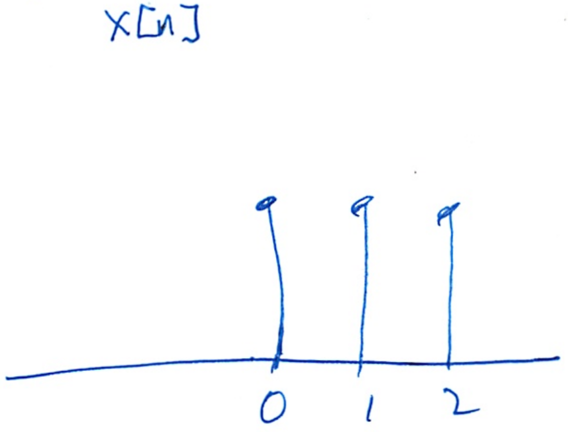
2/21
Lab Hrs

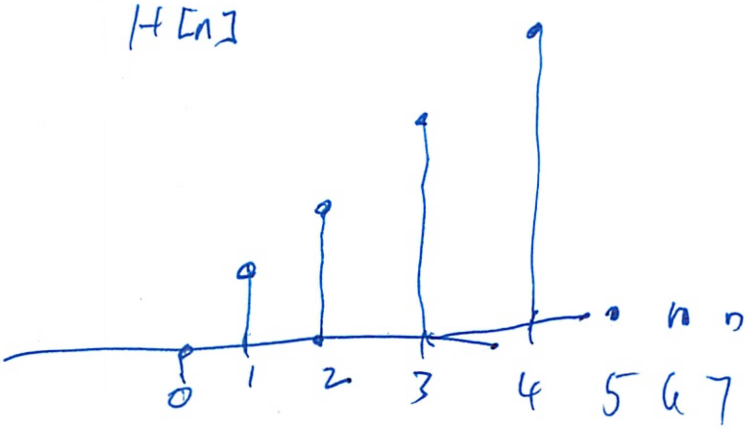
1.

lenght

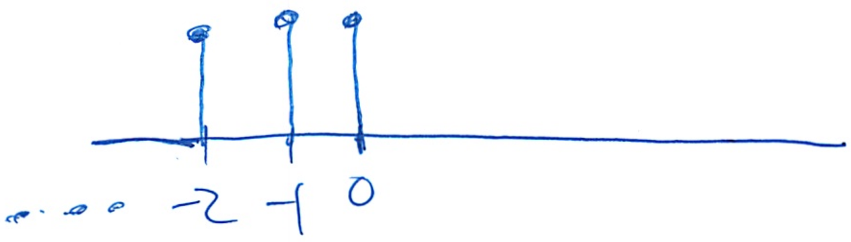


2.





flip
~~only~~ either one
and slide that



$y[0]$

$= 0 \cdot 1$

slide one ↳ $Y[1] = 1 \cdot 1 + 0 \cdot 1 = 1$

②

slide $y[2] = 2 \cdot 1 + 1 \cdot 1 + 0 \cdot 1 = 3$

slide $y[3] = 3 \cdot 1 + 2 \cdot 1 + 1 \cdot 1 + 0 \cdot 1 = 6$

$y[4] = 4 \cdot 1 + 3 \cdot 1 - - - = 10$

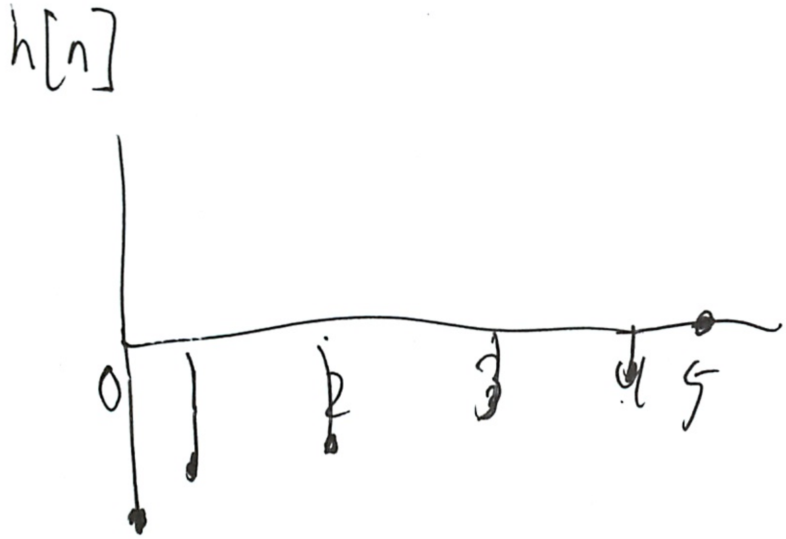
$y[5] = 11$ $\quad$ 10

11 $\quad$ 10

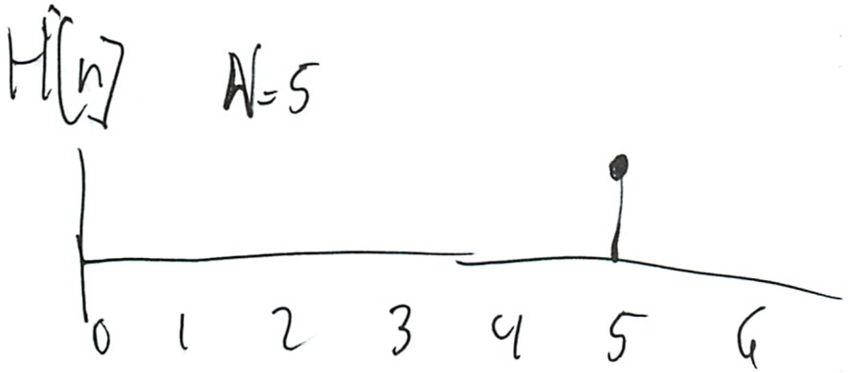
11 $\quad$ 10

Still it will not change?

Say $N=5$



Well δ = 1 at [0]
0 else where

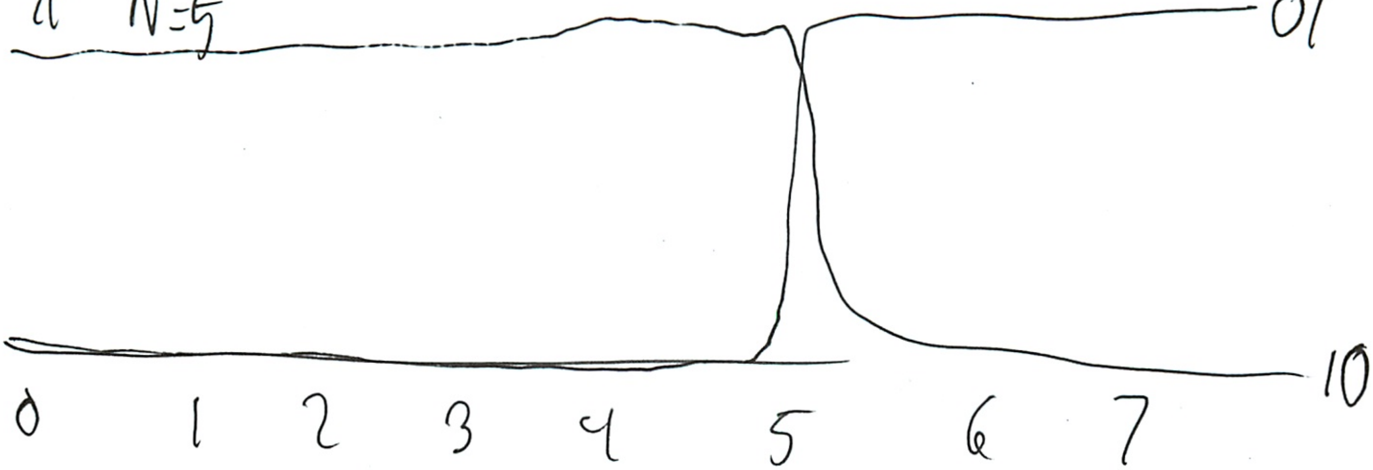


H means delay
But H describes channel response
Not input

But Then when we add input, what happens?

②

So if N=5



↑

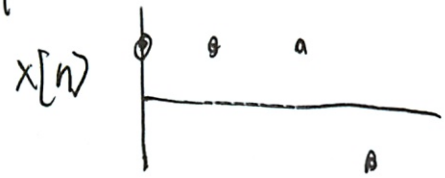
just where it crosses?

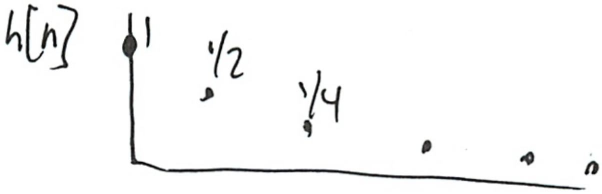
So lenghtens horizontally →

Submitted online

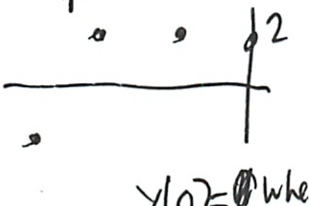
See Piazza

4





↳ Flip



y[0]= when this starts at 0

$Y[0] = 2 \cdot 1 = 2$

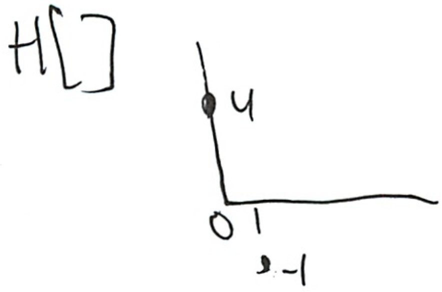
$Y[1] = 2 \cdot \frac{1}{2} + 2 \cdot 1 = 3$

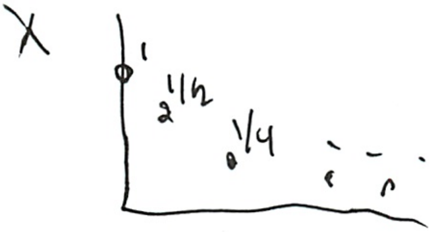
$Y[2] = 2 \cdot \frac{1}{4} + 2 \cdot \frac{1}{2} + 2 \cdot 1 = 3.5$ ← biggest

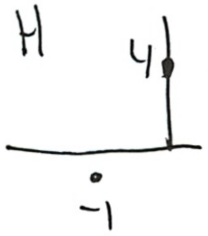
$y[3] = 2 \cdot \frac{1}{8} + 2 \cdot \frac{1}{4} + 2 \cdot 1/2 + -2 \cdot 1 = -1/4$

$Y[4] = 2 \cdot \frac{1}{16} + 2 \cdot \frac{1}{8} + 2 \cdot 1/4 + -2 \cdot 1/2 = 1 - \frac{7}{16} = \frac{9}{16}$

② Now test



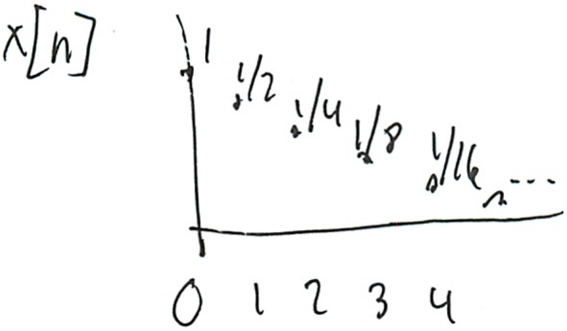


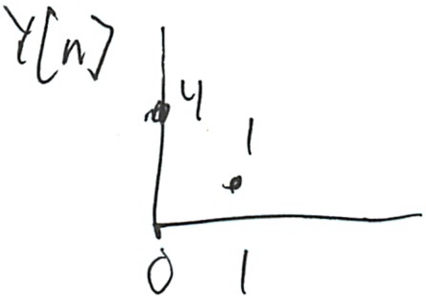


$Y[0] = 1 \cdot 4 = 4$

$Y[1] = \frac{1}{2} \cdot 4 + 1 \cdot -1 = 2 - 1 = 1$ ✓

5.





¿ Solve backwards

¿ deconvolve?

Usually get w/ unit sample

Let me try work backwards

$$4 = 1 \cdot h[0]$$

$$h[0] = 4$$

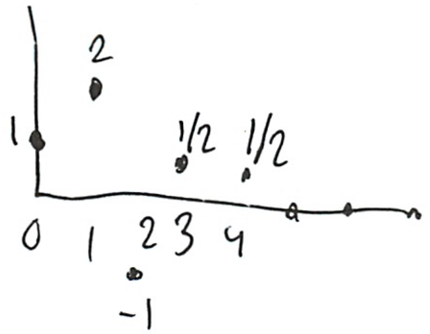
$$1 = \frac{1}{2} \cdot h[0] + 1 \cdot h[1]$$

$$1 = \frac{1}{2} \cdot 4 + 1 \cdot h[1]$$

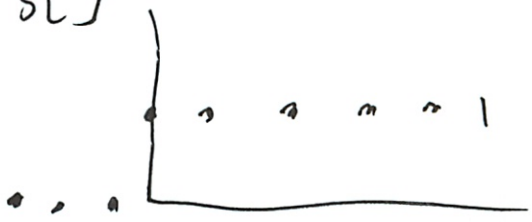
$$-1 = 1 \cdot h[1]$$

$$h[1] = -1$$

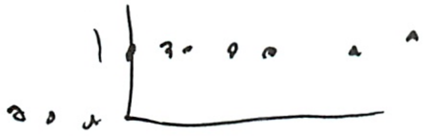
6. h[ ]

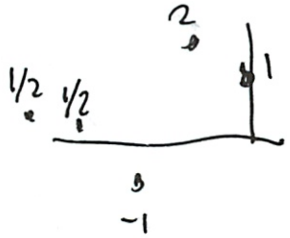


x[ ] = s[ ]



Y[n] = ?





$Y[0] = 1 \cdot 1 \quad = \quad 1$

$Y[1] = 1 \cdot 1 + 1 \cdot 2 \quad = 3$

$Y[2] = \quad '' \quad '' \quad + -1 \cdot 1 \quad = 2$

$Y[3] \quad '' \quad '' \quad '' \quad + 1 \cdot \frac{1}{2} = 2\frac{1}{2}$

$Y[4] = \quad '' \quad '' \quad '' \quad '' \quad + 1 \cdot \frac{1}{2} = 3$

$Y[5] \quad '' \quad '' \quad '' \quad '' \quad '' \quad = \quad ''$

↓ etc ↓

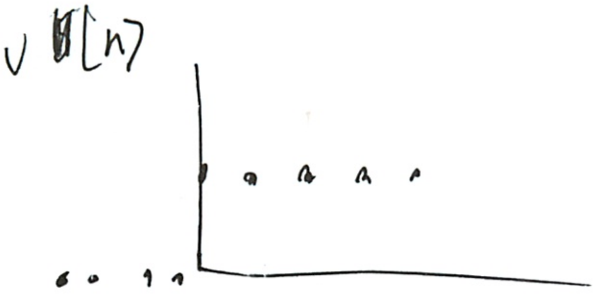
②

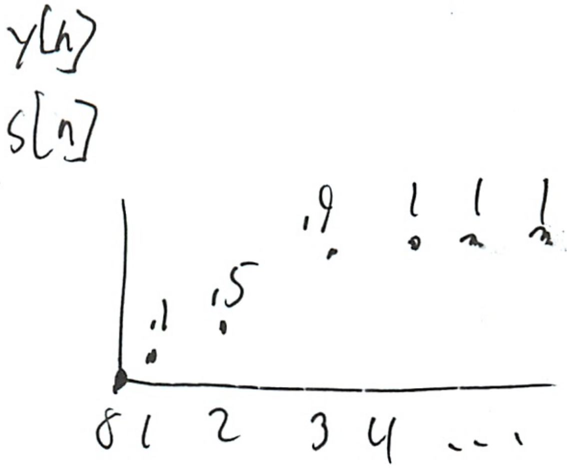
b) - So added

Ok

So $Y[\infty] = 1 \cdot 1 + 1 \cdot 2 + -1 \cdot 1 + 1 \cdot \frac{1}{2} + 1 \cdot \frac{1}{2} + 1 \cdot \underset{\uparrow 2}{h[5]} = 5$

7.





Want h[n]

$.1 = 1 \cdot h[0]$

$h[0] = .1$

$.5 = 1 \cdot .1 + 1 \cdot h[1]$

$.4 = h[1]$

$.9 = 1 + .1 + 1 \cdot .4 + 1 \cdot h[2]$

$.4 = h(2)$

$1 = .9 + 1 \cdot h[3]$

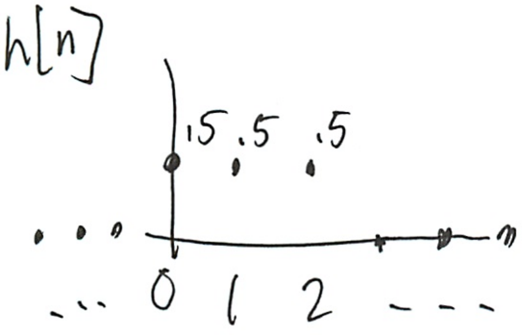
$h[3] = .1$

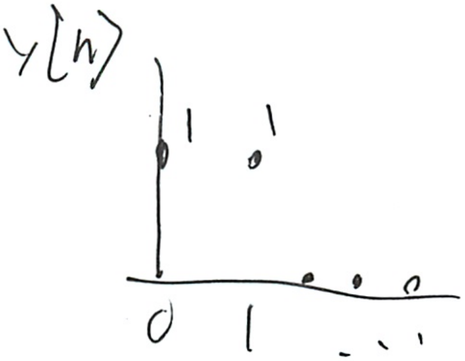
$1 = 1 + 1 \cdot h[4]$

$h[4] = 0$

$h[n \geq 4] = 0$

8.





Want x[n]

- same as other problem

$1 = x[0] \cdot .5$

$x[0] = 2$

$1 = 2 \cdot .5 + x[1] \cdot .5$

$x[1] = 0$

$0 = 2 \cdot .5 + 0 \cdot .5 + x[2] \cdot .5$

$x[2] = -2$

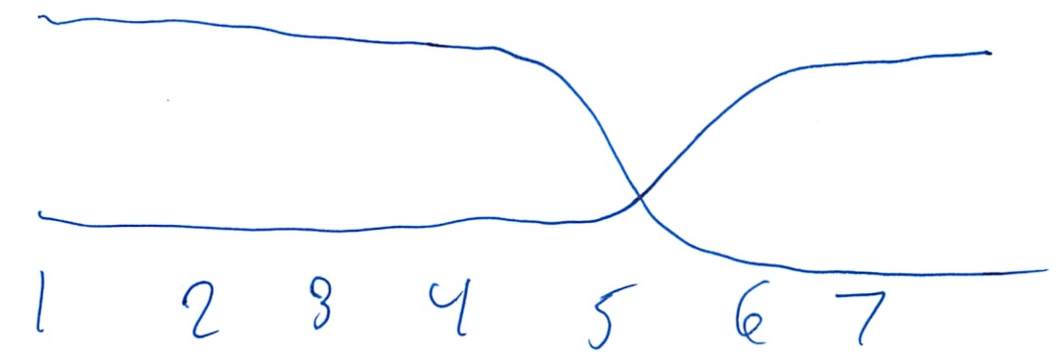
$0 = 2 \cdot .5 + 0 \cdot .5 + -2 \cdot .5 + x[3] \cdot 0$

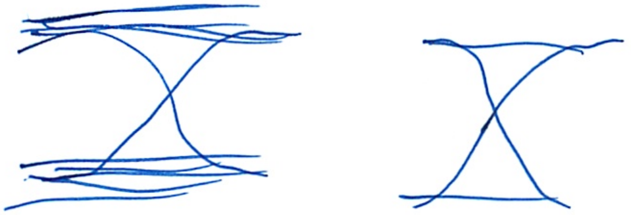
$x[3] = 0$

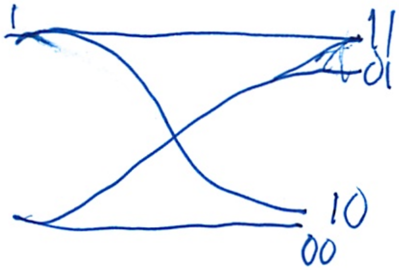
$\vdots$

P-Set 3 interview

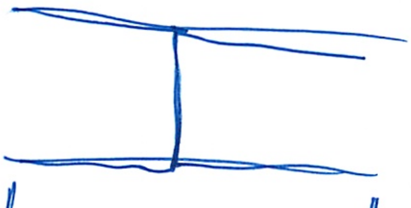
2/25





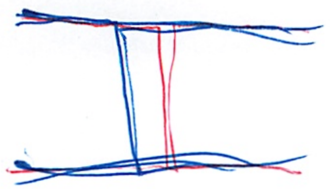


h[ ] - descibes channel



perfect channel
- instant

"Scale is $2 \cdot \frac{\text{sample}}{\text{bit}}$"



wrapping is only 1 sample unit - so would shift here

②

If delay was 10 - would see wrap

?divisable by n

For the +1 shift

Did $\geq$ when $>$ or something like that

Ringing - can't read

Slow -

+1 gets truncated

- has 0 up front
- can't deconvolve
- but when add noise
- the 0s are no longer discarded
- so upfront have some buffer
  - drop not only 0 but

$$-1e^{-5} \leq 0 \leq 1e^{-5}$$

Change based on noise

just pick someting - compare to max signal -

③ Like in PS 3 Task 1

Ringing ~~does~~ not really why making robust

key h[0]

- is significant is ringing channel
- ~~[illegible]~~ - noise does not effect much %-wise

Exam Thur

- Course org changed so previous exams not cover same
  - on prior year websites ~~[illegible]~~ accurate
- Should know tutorial problems

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 formula[1] 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).

[1] There 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]).

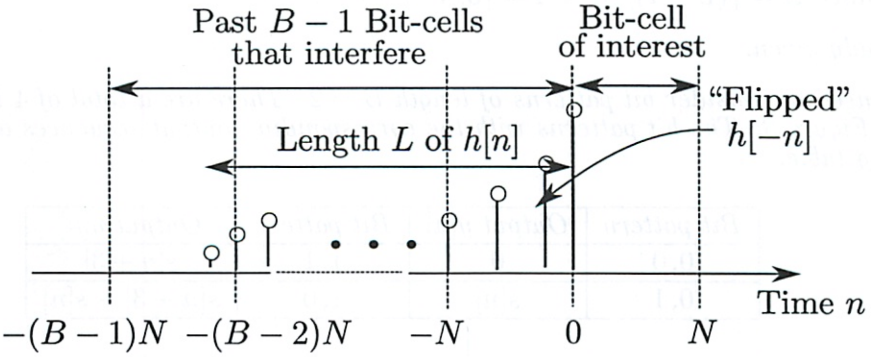


Figure 1: How many bit cells B do we need to consider?

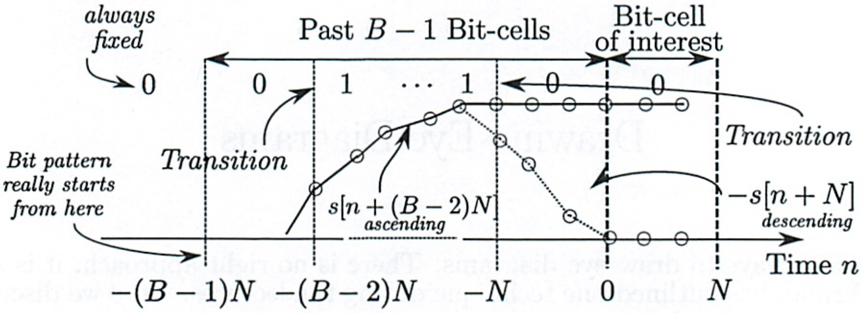


Figure 2: Each bit transition is associated with a step transition $s[n]$. If the bit transits from $0 \rightarrow 1$, it is an *ascending* transition. If bit transits $1 \rightarrow 0$, the transition is a *descending* one.

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 $1 \rightarrow 0$, draw[2] 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 $B = \lceil (L-1)/N \rceil + 1 = \lceil 3/4 \rceil + 1 = 2$.*

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	$s[n+3] - s[n]$

[2] I 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.

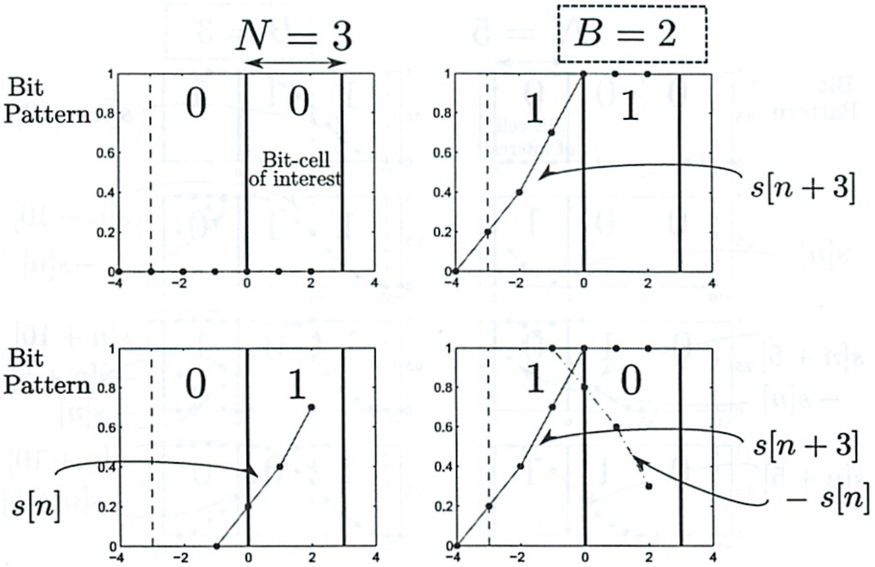


Figure 3: Example 1.

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 $B = \lceil (L-1)/N \rceil = \lceil 9/5 \rceil = 2$.*

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"[3] $s[n + (B-1)N]$.

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

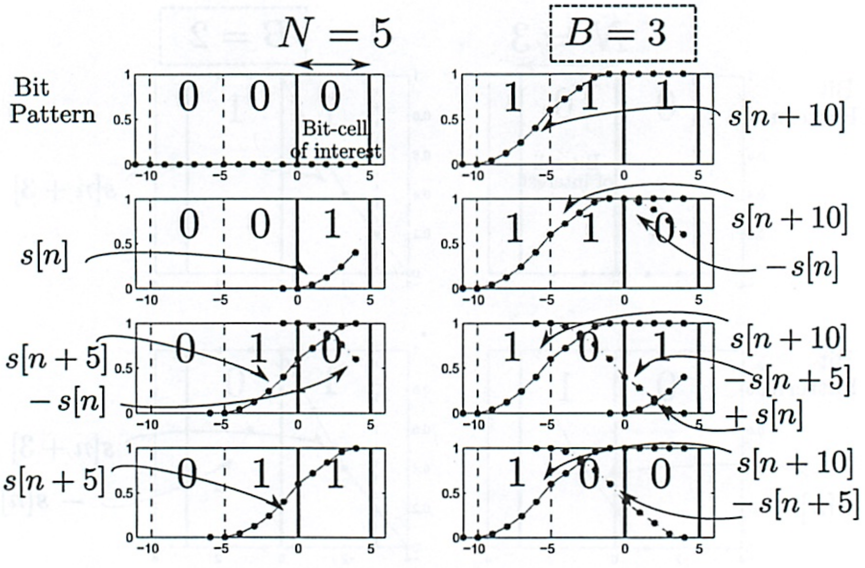


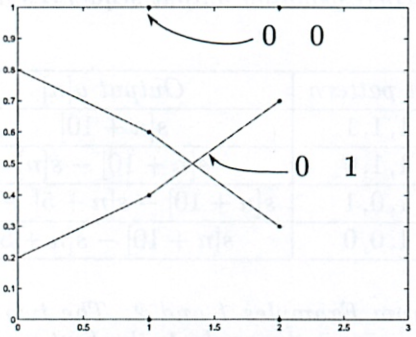
Figure 4: Example 2.

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

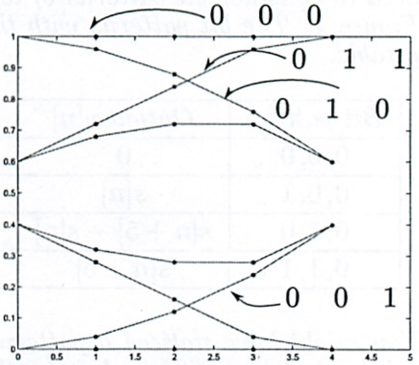
This means that we really only need to determine the left bit patterns; the right (flipped) patterns are obtained automatically by flipping the left outputs. This reduces the number of patterns we need to consider by half.

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.



Example 1



Example 2

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* *a string of 32 bits*	*(transition pts (multiples of N))* $4,5,6,7,8,11,12,14,$ $17,22,24,26,27,28,30,31$

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.

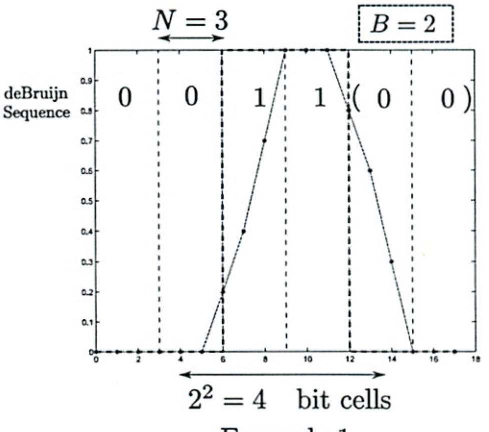
We can use deBruijn sequences to generate eye diagrams as follows:

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

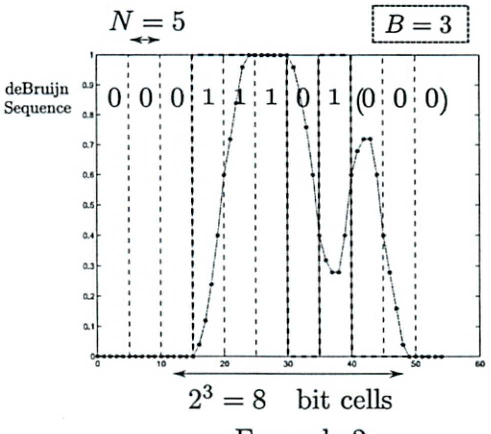
Step 2: Transmit this deBruijn sequence (using N samples per bit) and get channel output $y[n]$.

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.*

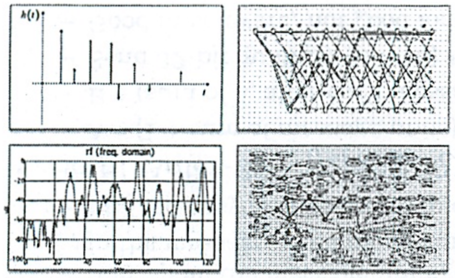


Example 1



Example 2

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.



INTRODUCTION TO EECS II

DIGITAL COMMUNICATION SYSTEMS

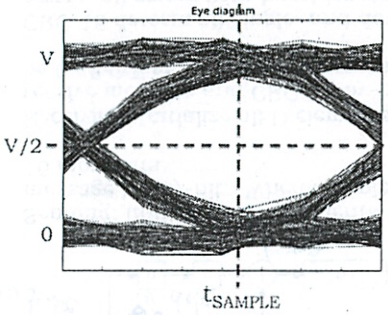
6.02 Spring 2011 Lecture #8

Quiz Thur Walker Gym
1 pg cribsheet

- Coping with errors using packets
- Detecting errors: checksums, CRC
- Hamming distance & single error correction
- (n,k) block codes

2/28

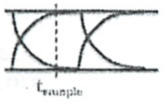
There's good news and bad news…



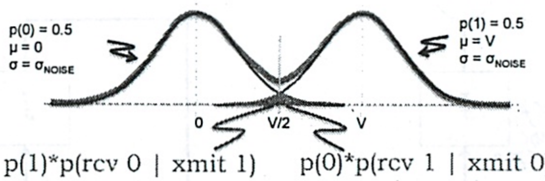
The good news: Our digital signaling scheme usually allows us to recover the original signal despite small amplitude errors introduced by inter-symbol interference and noise. An example of the digital abstraction doing its job!

The bad news: larger amplitude errors (hopefully infrequent) that change the signal irretrievably. These show up as bit errors in our digital data stream.

Bit Errors



Assuming a Gaussian PDF for noise and only 1-bit of inter-symbol interference, samples at t_{SAMPLE} have the following PDF:



want overlap to be smallest

We can estimate the bit-error rate (BER) using Φ, the unit normal cumulative distribution function:

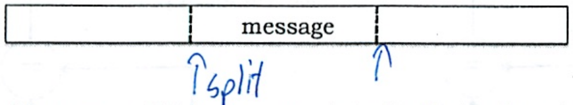
$$BER = \overbrace{(0.5)\Phi\left[\frac{V/2-V}{\sigma_{NOISE}}\right]} + \overbrace{(0.5)\left[1-\Phi\left[\frac{V/2-0}{\sigma_{NOISE}}\right]\right]} = \Phi\left[\frac{-V/2}{\sigma_{NOISE}}\right]$$

transmit 1 got 0

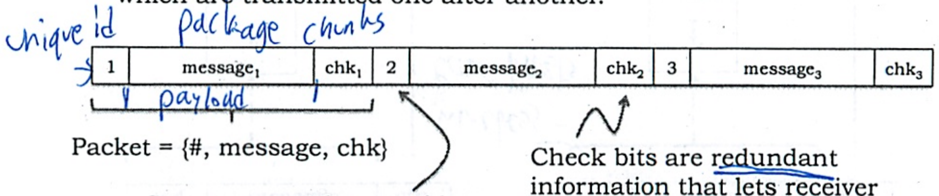
transmit 0 got 1

For a smaller BER, you need a smaller σ_{NOISE} or a larger V!

Dealing With Errors: Packets



To deal with errors, divide message into fixed-sized packets, which are transmitted one after another.



Packet = {#, message, chk}

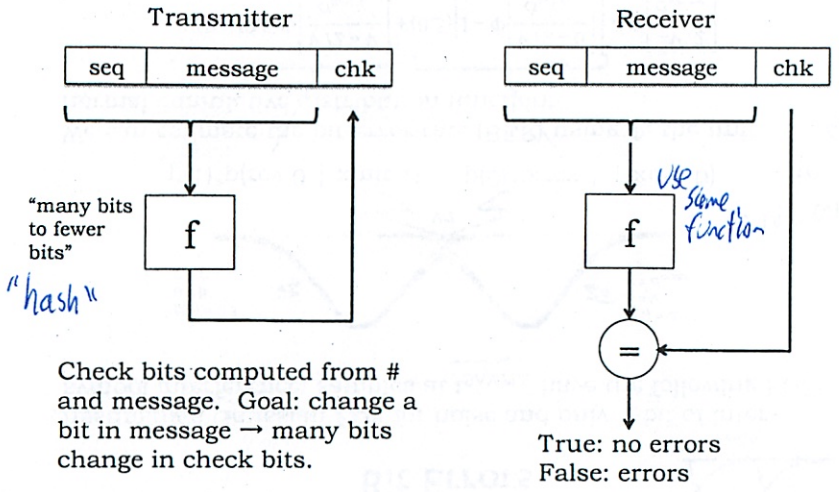
Sequence number provides unique identifier for each packet.

Check bits are redundant information that lets receiver verify # and message. Failure? Ask for packet to be resent.

but don't know # - ~~packet list~~ can't trust anything
So make list of good packets and ask for missing ones to be resent

Packet size: Too small → #/chk overhead is large
Too big → p(error) is larger, more to resend

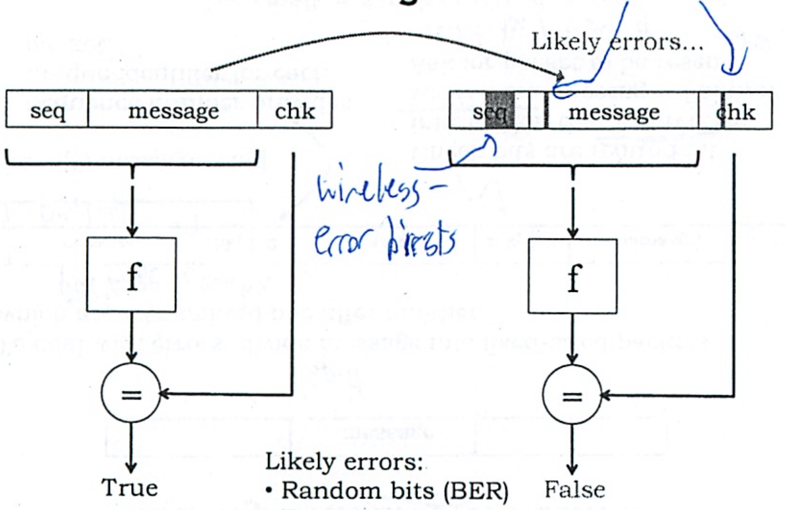
Check bits



Check bits computed from # and message. Goal: change a bit in message → many bits change in check bits.

Detecting Errors

Make sure two errors don't offset each other



Likely errors:

- Random bits (BER)
- Error bursts

err on side of calling good packets bad

Checksums

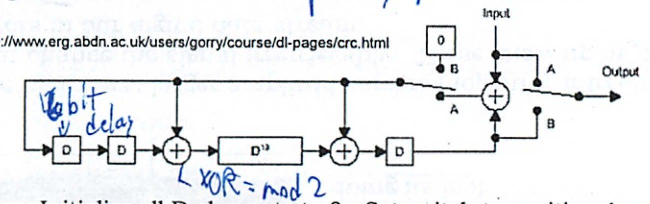
- Simple checksum
  - Add up all the message units, send along sum
  - Easy for two errors to mask one another offset each other
    - Some 0 bit changed to a 1; 1 bit in same position in another message unit changed to a 0... sum is unchanged
- Weighted checksum
  - Add up all the message units, each weighted by its index in the message, send along sum
  - Still too easy for two errors to offset one another
- Both! Adler-32 used in zip
  - A = (1 + sum of message units) mod 65521
  - B = (sum of A_i after each message unit) mod 65521
  - Send 32-bit quantity (B<<16) + A
  - Good in software, not good for short messages

Cyclical Redundancy Check

Example: CRC-16 finite fields, polynomial, etc

http://www.erg.abdn.ac.uk/users/gorry/course/dl-pages/crc.html

Cheap easy hardware



Sending: Initialize all D elements to 0. Set switch to position A, send message bit-by-bit. When complete, set switch to position B and send 16 more bits.

Receiving: Initialize all D elements to 0. Set switch to position A, receive message and CRC bit-by-bit. If correct, all D elements should be 0 after last bit has been processed.

CRC-16 detects all single- and double-bit errors, all odd numbers of errors, all errors with burst lengths < 16, and a large fraction ($1-2^{-16}$) of all other bursts.

This is foundation building on

many bits affect each bit

Approximate BER for common channels

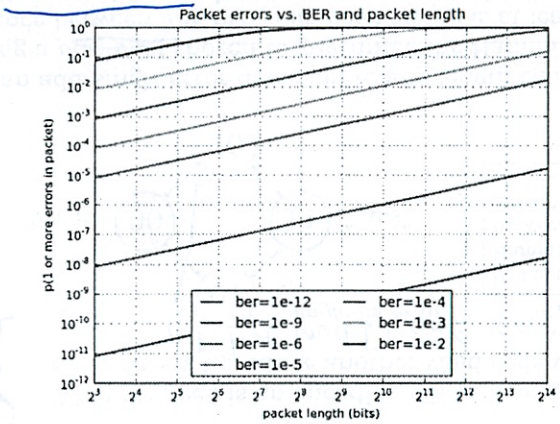
Channel type	Bandwidth	BER
Telephone Landline	2 Mbits/sec	10^{-4} to 10^{-6}
Twisted pair (differential)	1 Gbits/sec	$\leq 10^{-7}$ Ethernet
Coaxial cable	100 Mbits/sec	$\leq 10^{-6}$
Fiber Optics	10 Tbits/sec	$\leq 10^{-9}$
Infrared	2 Mbits/sec	10^{-4} to 10^{-6}
3G cellular	1 Mbits/sec	10^{-4}

Source: Rahmani, et al, *Error Detection Capabilities of Automotive Technologies and Ethernet – A Comparative Study*, 2007 IEEE Intelligent Vehicles Symposium, p 674-679

Very rough data

How Frequent is Packet Retransmission?

$$p(\text{1 or more errors}) = 1 - p(\text{no errors}) = 1 - (1 - \text{BER})^k$$



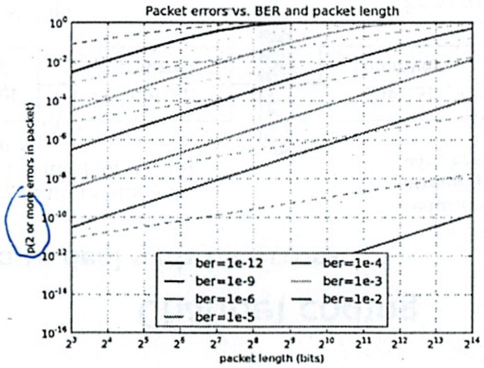
With 1kbyte packets and BER=1e-6, retransmit 1 every 100.

Implement Single Error Correction?

To reduce retransmission rate, suppose we invent a scheme that can correct single-bit errors and apply it to sub-blocks of the data packet (effectively reducing k). Does that help?

$$p(\text{2 or more errors}) = 1 - p(\text{no errors}) - p(\text{exactly one error})$$
$$= 1 - (1 - \text{BER})^k - k{*}\text{BER}{*}(1-\text{BER})^{k-1}$$

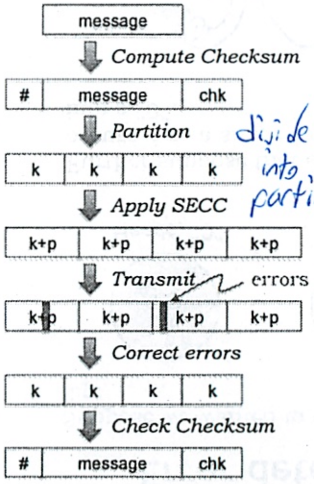
Use this often for hard drives



ECC - error correct.

Use p bit to correct

Digital Transmission using SECC



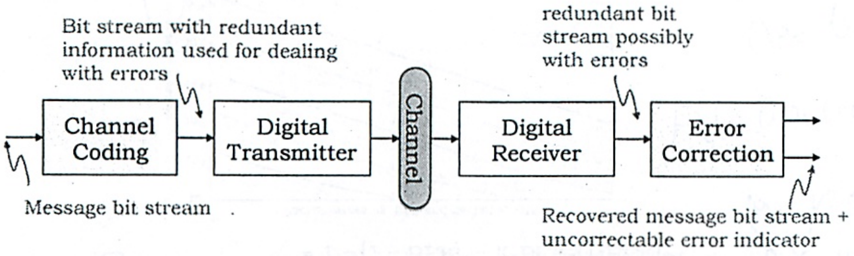
divide into partitions

- Start with original message
- Add checksum to enable verification of error-free transmission
- Apply SECC, adding parity bits to each k-bit block of the message. Number of parity bits (p) depends on code:
  - Replication: p grows as O(k)
  - Rectangular: p grows as O($\sqrt{k}$)
  - Hamming: p grows as O(log k)
- After xmit, correct errors
- Verify checksum, fails if undetected/uncorrectable error
- Deliver or discard message

Introduce end ~~k+1~~ bits on k-bit blocks

Channel coding

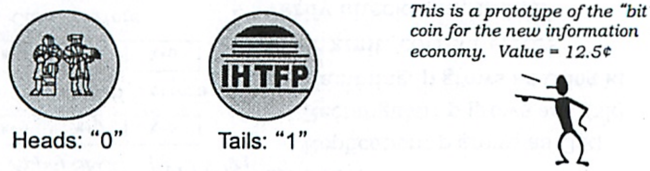
Our plan to deal with bit errors:


Bit stream with redundant information used for dealing with errors
redundant bit stream possibly with errors
Channel Coding
Digital Transmitter
Channel
Digital Receiver
Error Correction
Message bit stream
Recovered message bit stream + uncorrectable error indicator

We'll add redundant information to the transmitted bit stream (a process called channel coding) so that we can detect errors at the receiver. Ideally we'd like to correct commonly occurring errors, e.g., error bursts of bounded length. Otherwise, we should detect uncorrectable errors and use, say, retransmission to deal with the problem.

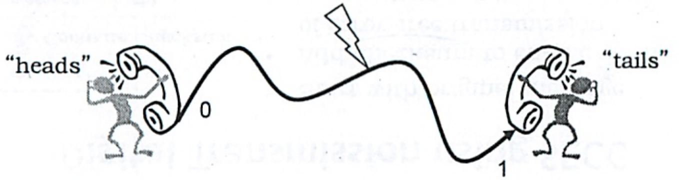
Error detection and correction

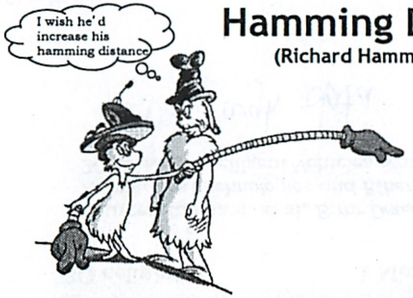
Suppose we wanted to reliably transmit the result of a single coin flip:



This is a prototype of the "bit" coin for the new information economy. Value = 12.5¢

Further suppose that during transmission a single-bit error occurs, i.e., a single "0" is turned into a "1" or a "1" is turned into a "0".





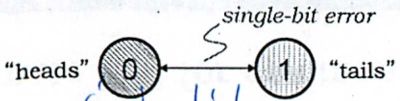
Hamming Distance

(Richard Hamming, 1950)

HAMMING DISTANCE: The number of digit positions in which the corresponding digits of two encodings of the same length are different

The Hamming distance between a valid binary code word and the same code word with single-bit error is 1.

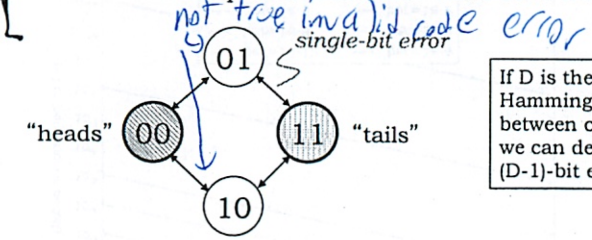
The problem with our simple encoding is that the two valid code words ("0" and "1") also have a Hamming distance of 1. So a single error changes a valid code word into another valid code word...



single bit error
Converts one value another

Error Detection

What we need is an encoding where a single-bit error doesn't produce another valid code word.



If D is the minimum Hamming distance between code words, we can detect up to (D-1)-bit errors

We can add single error detection to any length code word by adding a parity bit chosen to guarantee the Hamming distance between any two valid code words is at least 2. In the diagram above, we're using "even parity" where the added bit is chosen to make the total number of 1's in the code word even.

If more than 1 bit error - screwed!

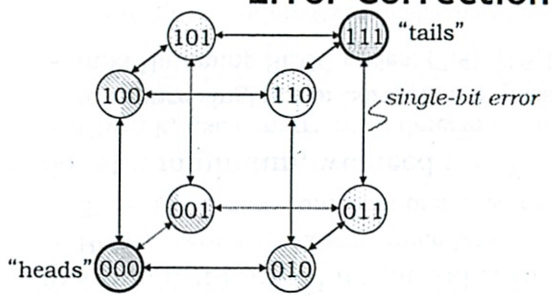
Parity check

- A parity bit can be added to any length message and is chosen to make the total number of "1" bits even (aka "even parity").
- To check for a single-bit error (actually any odd number of errors), count the number of "1"s in the received message and if it's odd, there's been an error.

```
0 1 1 0 0 1 0 1 0 0 1 1 → original word with parity
0 1 1 0 0 0 0 1 0 0 1 1 → single-bit error (detected)
0 1 1 0 0 0 1 1 0 0 1 1 → 2-bit error (not detected)
```

- One can "count" by summing the bits in the word modulo 2 (which is equivalent to XOR'ing the bits together).

Error Correction



If D is the minimum Hamming distance between code words, we can correct up to $\left\lfloor \frac{D-1}{2} \right\rfloor$ bit errors

By increasing the Hamming distance between valid code words to 3, we guarantee that the sets of words produced by single-bit errors don't overlap. So if we detect an error, we can perform *error correction* since we can tell what the valid code was before the error happened.

- Can we safely detect double-bit errors while correcting 1-bit errors?
- Do we always need to triple the number of bits?

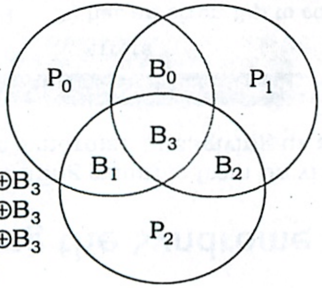
Single Error Correcting Codes (SECC)

Basic idea:

- Use multiple parity bits, each covering a subset of the data bits.
- No two message bits belong to exactly the same subsets, so a single error will generate a unique set of parity check errors.

Modulo-2 addition, aka XOR

$P_0 = B_0 \oplus B_1 \oplus B_3$
$P_1 = B_0 \oplus B_2 \oplus B_3$
$P_2 = B_1 \oplus B_2 \oplus B_3$



Suppose we check the parity and discover that P1 and P2 indicate an error?
bit B2 must have flipped

What if only P2 indicates an error?
P2 itself had the error!

Hamming single error bit code

Checking the parity

- Transmit: Compute the parity bits and send them along with the message bits
- Receive: After receiving the (possibly corrupted) message, compute a syndrome bit (E_i) for each parity bit. For the code on previous slide:

Syndrome bits

$$\left(\begin{array}{l} E_0 = B_0 \oplus B_1 \oplus B_3 \oplus P_0 \\ E_1 = B_0 \oplus B_2 \oplus B_3 \oplus P_1 \\ E_2 = B_1 \oplus \underline{B_2} \oplus B_3 \oplus P_2 \end{array} \right)$$

So for each combo. some is missing

- If all the E_i are zero: no errors! ~~missing~~ - so flip that one
- Otherwise the particular combination of the E_i can be used to figure out which bit to correct.

(did not pay close attention last 60 min)

Using the Syndrome to Correct Errors

Continuing example from previous slides: there are three syndrome bits, giving us a total of 8 encodings.

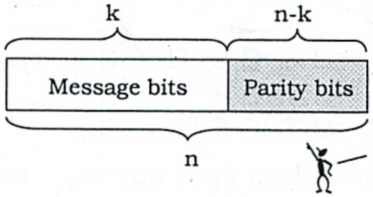
$E_2E_1E_0$	Single Error Correction
0 0 0	No errors
0 0 1	P0 has an error, flip to correct
0 1 0	P1 has an error, flip to correct
0 1 1	B0 has an error, flip to correct
1 0 0	P2 has an error, flip to correct
1 0 1	B1 has an error, flip to correct
1 1 0	B2 has an error, flip to correct
1 1 1	B3 has an error, flip to correct

What happens if there is more than one error?

The 8 encodings indicate the 8 possible correction actions: no errors, error in one of 4 data bits, error in one of 3 parity bits

(n,k,d) Systematic Block Codes

- Split message into *k*-bit blocks
- Add (*n-k*) parity bits to each block, making each block *n* bits long.



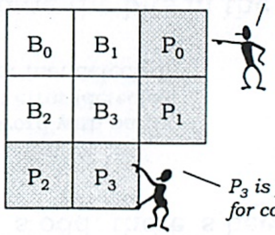
The entire block is called a "code word" and this is an (n,k) code.

- Often we'll use the notation (n,k,d) where d is the minimum Hamming distance between code words.
- The ratio k/n is called the *code rate* and is a measure of the code's overhead (always ≤ 1, larger is better).

A simple (8,4,3) code

Idea: start with rectangular array of data bits, add parity checks for each row and column. Single-bit error in data will show up as parity errors in a particular row and column, pinpointing the bit that has the error.

P_0 is parity bit for row #1



B_0	B_1	P_0
B_2	B_3	P_1
P_2	P_3	

P_3 is parity bit for column #2

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

Parity for each row and column is correct ⇒ no errors

Parity check fails for row #2 and column #2 ⇒ bit B_3 is incorrect

Parity check only fails for row #2 ⇒ bit P_1 is incorrect

Can you verify this code has a Hamming distance of 3?

How many parity bits to use?

- Suppose we want to do single-bit error correction
  - Need unique combination of syndrome bits for each possible single bit error + no errors
  - n-bit blocks → n possible single bit errors
  - Syndrome bits all zero → no errors
- Assume n-k parity bits (out of n total bits)
  - Hence there are n-k syndrome bits
  - $2^{n-k} - 1$ non-zero combinations of n-k syndrome bits
- So, at a minimum, we need $n \le 2^{n-k} - 1$
  - Given k, use constraint to determine minimum n needed to ensure single error correction is possible
  - (n,k) Hamming SECC codes: (7,4) (15,11) (31,26)

The (7,4) Hamming SECC code is shown on slide 19, see the Notes for details on constructing the Hamming codes. The clever construction makes the syndrome bits into the index needing correction.

6.02 Recitation
Exam 1 Review

3/1

Cheat sheet double sided

Topics

1. Information Entropy
2. Huffman Code
3. LZW code
4. LTI Linear Time Invarient
   - Unit sample signal
   - Convolution
   - deconvolution
5. Eye diagrams
6. Misc
   - 8b/10b
   - Syncronization

(I think I did well in class - just need to refresh my mind)

Say have prob $\frac{1}{6}$ $\frac{1}{6}$ $\frac{1}{3}$ $\frac{1}{9}$ $\frac{2}{9}$

So weighted avg the logs of the probabilities

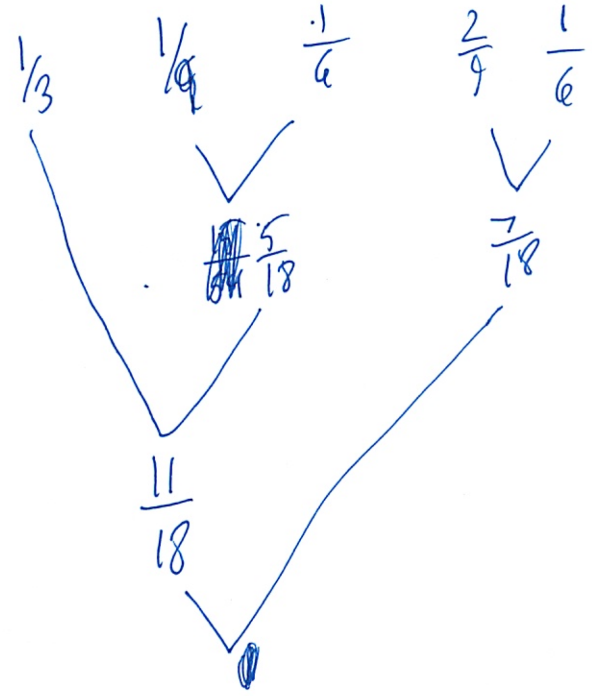
$$\frac{1}{6} \cdot \log_2 6 + \frac{1}{6} \log_2 6 + \frac{1}{3} \log_2 3 + \frac{1}{9} \log_2 9 + \frac{2}{9} \log_2 \frac{9}{2}$$

$$\approx 2.1$$

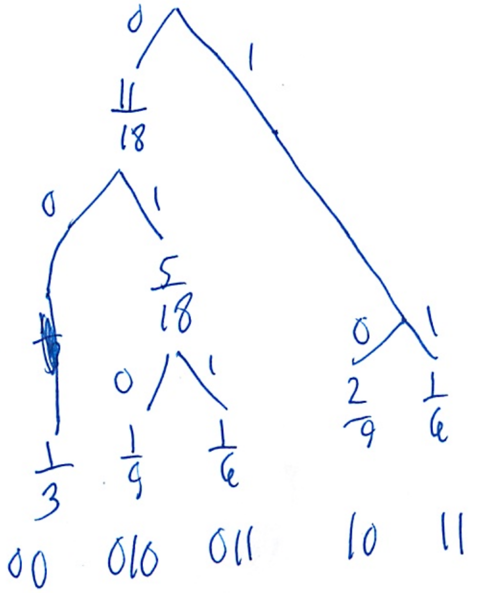
This is the minimum # bits per symbol for an average (w/ the prob dist) of each character

②

Now make a Huffman tree



Now assign code



Avg bit lenght

$2 + \frac{5}{18} \cdot 1$

so is weighted by symbol prob

③

LZW

Message: Once Upon A Time A Cat ate The Dog

Coding

```
String = *
Symbol = *

Get next symbol
    String + Symbol ∈? Table
        Yes: String = ~~Symbol~~ String + Symbol
        No: Insert String + Symbol
            Output Table(string)
            String = Symbol

Output Table(string)
```

Table

First some entries w/ alphabet preloaded
But we will simplify

A=1
~~B~~C=2
D=3
E=4
G=5
H=6

I=7
M=8
N=9
O=10
P=11

T=12
U=13

4

Once upon a time cat ate the dog

↓

	Output	Table
Symbol = O		
String = *	*	
↓		
String = O	10	ON = 14
Symbol = N		
↓		
String = N		
→ Move cursor		NC = 15
Symbol = C	9	
↓		
String = C		

Now <u>decode</u>

* - Do nothing

10 = O

9 = N

↳ add ON to table

If get something where don't know the last letter then it is the **first** letter of the string

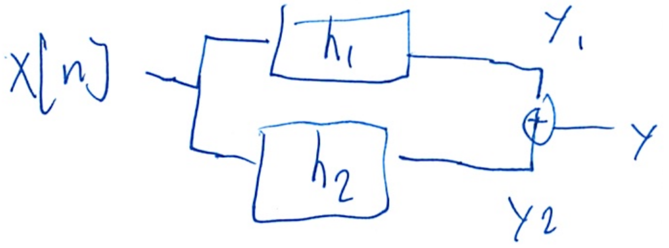
ZDC* → the * = Z

Algorithms exploit the structure of a problem
- the 'invariants' behind it

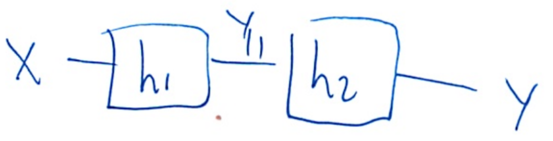
Convolution

$$y[n] = \sum_{k=-\infty}^{\infty} x[n-k]h[k]$$

$$x[n] = \frac{1}{h[0]}(y[n] - \dots)$$



$$y[n] = y_1[n] + y_2[n] \;; \quad x \cdot h_1 + x \cdot h_2 = x \cdot (h_1 + h_2)$$



$$\begin{aligned} y &= y_1 \cdot h_2 \\ &= (x \cdot h_1) \cdot h_2 \\ &= x(h_1 \cdot h_2) \end{aligned}$$

6)

Eye Diagrams

N = samples/bit

M = Memory of h, depends on h, your channel

$B = \left[\frac{m}{N} \right] + 1$

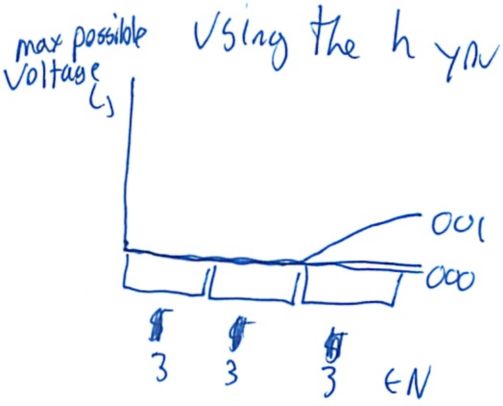
Find each possible combo

N = 3
M = 5

So B = 3

000
001
010
⋮

Using the h you have



Voltage over time

6.02 Review Session

- Format : Tutorial Problems

1. Bunch of people guessing 3 bit #

a) Alice told it is odd. How much bits of info did she get.

So 3 bit #

0 → 7

So half is even, odd

So 8 had

4 have

↙ inverse of prob

Been given $\log_2\left(\frac{8}{4}\right)$ information

$= 1$ bit

b) Bob told not multiple of 3

So not 3, 6

0 is included as well

$\log_2\left(\frac{8}{5}\right) \approx .6$

↑ remaining #

c) Charlies told has 2 11 in binary

②

000

010

001

100 $\qquad \log_2\left(\frac{8}{3}\right) \approx 1.45$

011 ←

101

110 ←

111 ←

d) Told all 3 above

- told exactly what # it is
- So 3 bits of info

$$\log_2\left(\frac{8}{1}\right) = 3$$

---

2. know x is 8-bit binary #

know y differs in 1 bit

How much info given about x ?

$$\log_2\left(\frac{\overset{\downarrow \text{ all}}{256}}{8}\right) = 5 \text{ bits}$$

↑ the ~~know #~~

# of possibilities it still can be

③

3. ~~Record~~ Recieved E symbols

Express # as encoded for A symbols

So if get stream of aaaaa... how many bits to send?

Output	Table
1	256 aa
256	257 aaa
257	258 aaaa
258	⋮
⋮	

So $\frac{E(E+1)}{2}$ ← I am bad at figuring this out

---

4. Decoder

Will give pseudo code on exam

Just do in reverse

Went over in recitation

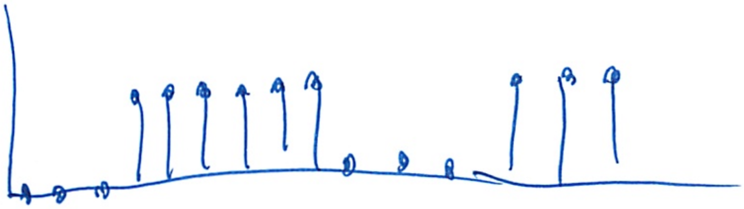
④

Its just the previous block plus first letter of next block

Digitial Signaling

# of samples per bit

Look at min width of either 1 or 0 seq

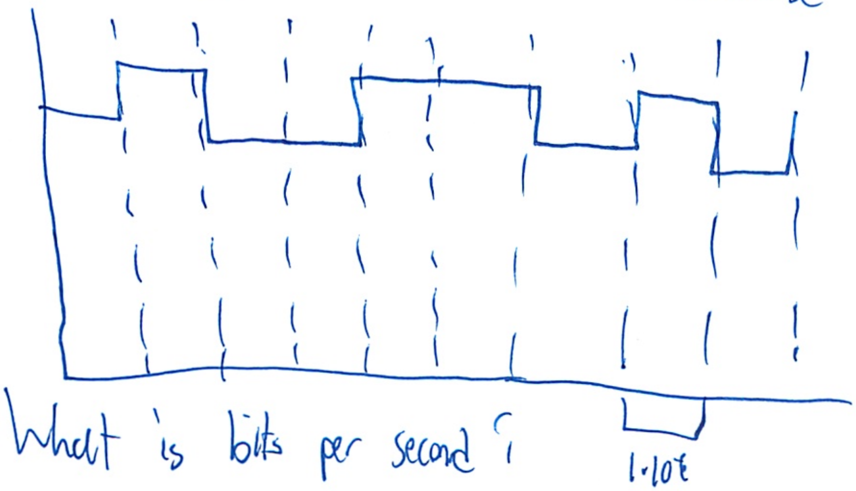


Is 3

So look in the middle of every interval of 3

If transition is in middle - decrease # samples per bit

2.

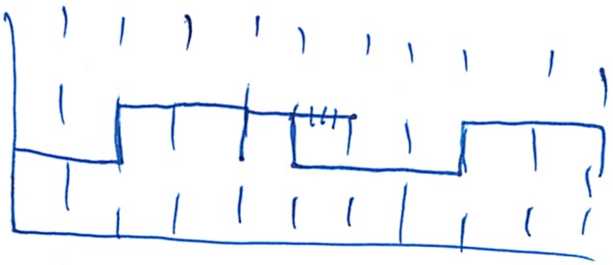


What is bits per second?

$$\frac{1}{1\cdot10^{-6}} = 1\cdot10^{+6} \text{ bits/sec}$$

5

b)



I think it is 2 samples/bit

but here does not work!

So fall down to 1 sample/bit

8b10

lets you know where packet begins

not clock recovery

- that was that speed up/slow down reading
- align sampling interval
- not very good at long seq of 0s, 1s
    - errors accumulate

8b/10 re-encods so transitions in there

TA does not know algorithm to go to 10 bit

- Look at wikipedia

⑥

LTI System

How to check is a LTI system?

$$y[n] = 2x[n] + 3$$

$$\begin{aligned}
x_1[n] &\rightarrow y_1[n] = 2x_1[n] + 3 \\
x_2[n] &= y_2[n] = 2x_2[n] + 3 \\
x_3[n] &= x_1[n] + x_2[n] \rightarrow y_3[n] = 2x_3[n] + 3 \\
&\qquad\qquad = 2(x_1[n] + x_2[n]) + 3 \\
&\qquad\qquad \neq y_1[n] + y_2[n]
\end{aligned}$$

So <u>not</u> a linear system

(Need clarification on this)

Check for time invarient

$$\begin{aligned}
x[n-n_0] \rightarrow y_{out} &= 2x[n-n_0] + 3 \\
&= y[n-n_0]
\end{aligned}$$

<u>is</u> time invarient

7

$$y[n] = n\,x[n]$$

$$x_1[n] \rightarrow y_1[n] = n\,x_1[n]$$

$$x_2[n] \rightarrow y_2[n] = n \cdot x_2[n]$$

$$x_3[n] = x_1[n] + x_2[n]$$

$$y_3[n] = n\left(x_1[n] + x_2[n]\right)$$

$$= y_1[n] + y_2[n]$$

$\underline{\text{is}}$ linear system

$$x[n-n_0] \rightarrow y_{out} = n\,x[n-n_0]$$

$$\neq y[n-n_0]$$

$$= (n-n_0)\,x[n-n_0]$$

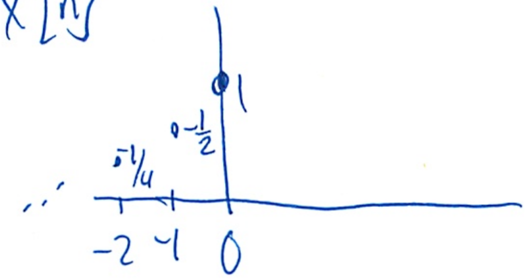
$\underline{\text{not}}$ time invarient

8

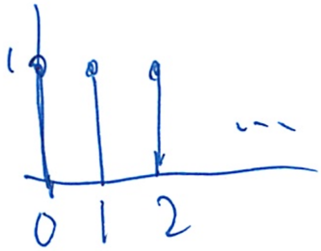
$x[n] = 2^n u[n]$ input to system

$h[n] = u[n]$ unit response to system

$x[n]$



$h[n]$



So what is output? $y[n]$

Use convolution

$$y[n] = \sum_{k=-\infty}^{+\infty} x[k]h[n-k]$$

note that $x[k] \neq 0$, $k < 0$

$h[n-k] \neq 0$, $n-k > 0$

$n > k$

$$y[n] = \sum_{k=-\infty} x[k]\, h[n-k]$$

⑨ When $n > 0$

$$Y[n] = \sum_{k=-\infty}^{0} x[k]$$

$$= \sum_{k=\infty}^{0} 2^k$$

$$= 2^0 + 2^{-1} + 2^{-2} + \dots + 2^{-\infty}$$

So geometric sum

$$\frac{2^0(1-2^{-\infty})}{1-(2^{-1})} = 2$$

When $k < n < 0$

$$Y[n] = \sum_{k=-\infty}^{n} x[k] = \sum_{k=-\infty}^{n} 2^k$$

$$= \frac{2^n(1-\cancel{2^{\infty}})}{1-2^{-1}}$$

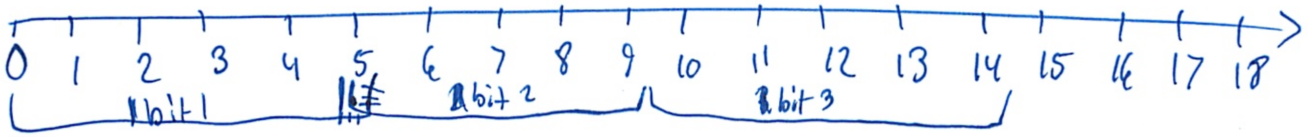
$$= 2^{n+1}$$

(I think same as I do it but graphically)

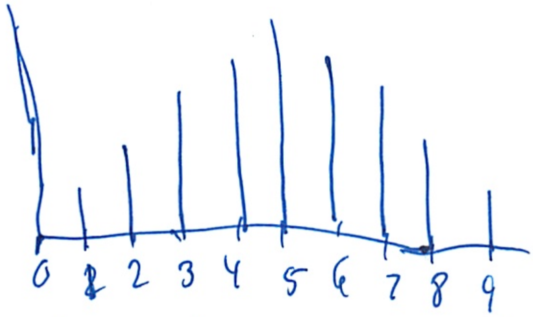
10

Eye Diagrams

1. 5 samples / bit



h[ ] =



Want to know how many bits interfere w/ our current bit?
L "eye size"

Look at eye diagram

- don't need to know how to generate

8 lines

So 2^3, so 3 bits in there
L current
the two previous bits

Can also find eye size with h and samples / bit

(ii)

First bit no ISI

2nd bit yes 2 bits

3rd bit full ISI

So do all pattens of 3 bits

000
001
010
100
011
101
110
111

(I think this is confusing me more)

Do the convolution for each

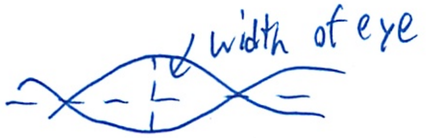
$$\begin{aligned} y[12] = {} & x[12]\,h[0] + \\ & x[11]\,h[1] + \\ & \vdots \\ & x[0]\,h[12] \end{aligned}$$

how get each of 8 calues

He did weird starting index

12

In this class the max possible is 010


width of eye