MASSACHUSETTS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

6.02 Introduction to EECS II
Spring 2011

Quiz 1

Name Michael Plasmeier	*Score* 76

☒	10a	Devavrat Shah	24-402
□	11a	Devavrat Shah	24-402
□	12n	Fabian Lim	38-166
□	1p	John Sun	38-166
□	2p	John Sun	38-166

Please write your answers legibly in the spaces provided. You can use the backs of the pages if you need extra room for your answer or scratch work. Make sure we can find your answer!

You can use a calculator and one 8.5" x 11" cribsheet.

Partial credit will only be given in cases where you show your work and (very briefly) explain your approach.

Problem	Score	
#1 (30 points)	25	g.s.
#2 (20 points)	15	CS
#3 (50 points)	36 Yu	

Problem 1. Information, Entropy and Huffman Codes (30 points)

There's a weekly surprise party at a local independent living group with an equal probability that the event will happen on any of the seven days.

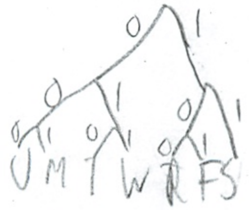
(A) (3 points) You learn that party won't be on the weekend, i.e., not Saturday or Sunday. Give an expression for the number of bits of information you have received.

Expression for number of bits of information received: $\log_2\left(\frac{7}{5}\right)$ ✓

↑# choices left

$\approx .48$

(B) (4 points) Give an expression for the expected length in bits of a Huffman encoding of a message that lists the day of the party for each week of the 52-week year, i.e., a message consisting of 52 variable-length symbols, where each day is encoded separately using the Huffman code. The choice for each week is independent of the choices for other weeks.



Expression for expected length of message in bits: $52\left(\frac{6}{7}\cdot 3 + \frac{1}{7}\cdot 2\right)$ ✓

$\frac{6}{7}\cdot$

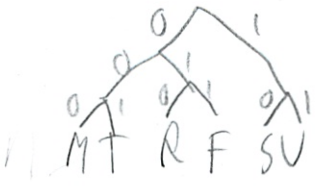
≈ 148.57

Examining the historical record, you discover that the probabilities for party days aren't in fact equal – weekends are very popular and the party is never held on Wednesday when 6.02 psets are due. You prepare the following table showing the updated probabilities, which should be used when answering the following questions.

day	*Mon*	*Tue*	*Wed*	*Thu*	*Fri*	*Sat*	*Sun*
p(day)	0.125 1/8	0.125	0	0.125	0.125	0.25 1/4	0.25
$log_2(1/p)$	3	3	--	3	3	2	2
$p*log_2(1/p)$	0.375	0.375	--	0.375	0.375	0.5	0.5
Encoding from part (C)	000	001	--	010	011	10	11 ✓

(C) (6 points) Using the updated probabilities, create a variable-length Huffman code for sending messages listing party days. Note that no code is required for Wednesday. Please enter the encoding for each day in the last row of the table above.

Fill in last table row



(D) (4 points) Compute the expected length in bits to encode message containing one day using your code from part (C). Please give a numeric answer.

$\frac{4}{8} \cdot 3 + \frac{2}{4} \cdot 2$

Expected length in bits: 2.5 ✓

(E) (4 points) Using the updated probabilities, compute the entropy of the underlying probability distribution. Please give a numeric answer. Hint: much of the computation has already been performed for you!

$\sum_{i=1}^{n} P(x_i) \log_2 \left(\frac{1}{P(x_i)} \right)$

$.375 \cdot 4 + .5 \cdot 2$

Entropy: 2.5 ✓

(F) (4 points) By changing the encoding scheme (say, by encoding pairs of days), would it be possible to improve the expected length of messages? Briefly explain why or why not.

Brief explanation

No, since our encoding is at entropy. ✓ There is no way to improve without ambiguity

(G) (5 points) A phone call from a friend causes you to revise the probabilities for the coming week as follows:

TA: floor + ceiling functions   Prof: For transmitting new probability

day	Mon	Tue	Wed	Thu	Fri	Sat	Sun
p(day)	0.1	0.1	0	0.1	0.1	0.6	0
$log_2(1/p)$	3.322	3.322	--	3.322	3.322	0.737	--
$p*log_2(1/p)$	0.332	0.332	--	0.332	0.332	0.442	--

I do not get this qv at all

How many bits of information did the phone call deliver? Please give a numeric answer.

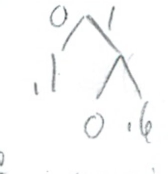
new probabilities delivered

Bits of information from phone call: 8 ✗ -5

Need to transmit 6 days since know Wed never party

- So person on phone does not mention

.1, .1, .1, .1, .6, 0



1+1+1+1+2+2 = 8

Assume tree code transmitted sepertly

Prof: There is a straight forward way, you don't seem aware of not log([illegible])

Problem 2. LZW compression (20 points)

An 8-character message was encoded using the LZW encoder whose pseudo-code is shown below:

Should have put decode on

```
STRING = get input symbol
WHILE there are still input symbols DO
   SYMBOL = get input symbol   first
   IF STRING + SYMBOL is in the string table THEN
       STRING = STRING + SYMBOL   appended
   ELSE
       output the code for STRING   for just string
       add STRING + SYMBOL to the string table
       STRING = SYMBOL
   END
END
output the code for STRING
```

When the encoding process was complete the following additions had been made to the string table:

Should have tried

table[256] = ho
table[257] = oh
table[258] = hoh
table[259] = hoho

decode-ish

(A) (10 points) What was the original 8-character message?

Original message: hohohoho ✓

(B) (10 points) Recall that the encoder only sends indices into the string table. What indices did the encoder send? Hint: everything can be figured out from the string entries and their order. The index of 'h' is 104 and of 'o' is 111.

Indices sent by encoder: 104, 111, 256, ~~259~~

h o ho hoho

-5

258, 111

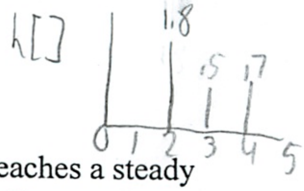
¡ don't have 259 yet!

104
111 h
ho -> 256 ho
256 257 oh
259 hoho -> 258 hoh
259 hoho

Problem 3. LTI Models for Communication Channels (50 points)

Consider a communications channel *C1* that is accurately modeled as a noise-free linear time invariant system with the following causal unit sample response:

$h_{C1}[0]$	$h_{C1}[1]$	$h_{C1}[2]$	$h_{C1}[3]$	$h_{C1}[4]$	$h_{C1}[\geq 5]$
0.0	0.0	1.8	0.5	0.7	0.0



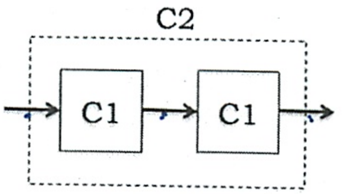
(A) (4 points) The unit step response for this channel, $s_{C1}[n]$, eventually reaches a steady state value v. What is v and what is the smallest k such that $s_{C1}[k] = v$?

the step

Steady state value v: 3 ✓

Smallest k: 4 ✓

(B) (10 points) Suppose we built a communications channel *C2* composed of two C1 channels connected in series:



Series → Multiply

Please fill in the following table, giving the first 10 values of the unit sample response for the C2 channel.

So do for 1 system - and multiply

not unit step!

Fill in table

output ↓

$h_{C2}[0]$	$h_{C2}[1]$	$h_{C2}[2]$	$h_{C2}[3]$	$h_{C2}[4]$	$h_{C2}[5]$	$h_{C2}[6]$	$h_{C2}[7]$	$h_{C2}[8]$	$h_{C2}[9]$
0	0	3.24	5.29	9	9	9	9	9	9

-8

One system 0 0 1.8 2.3 3 3 3 3 3 3

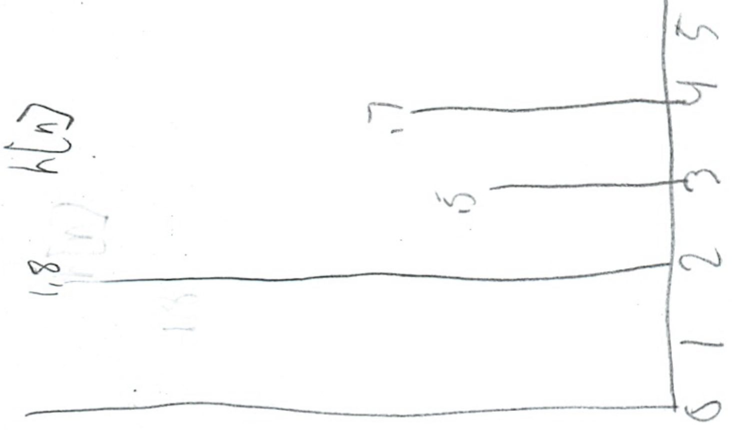
0 0 0 0 3.24 1.8 2.27 .7 .49 0

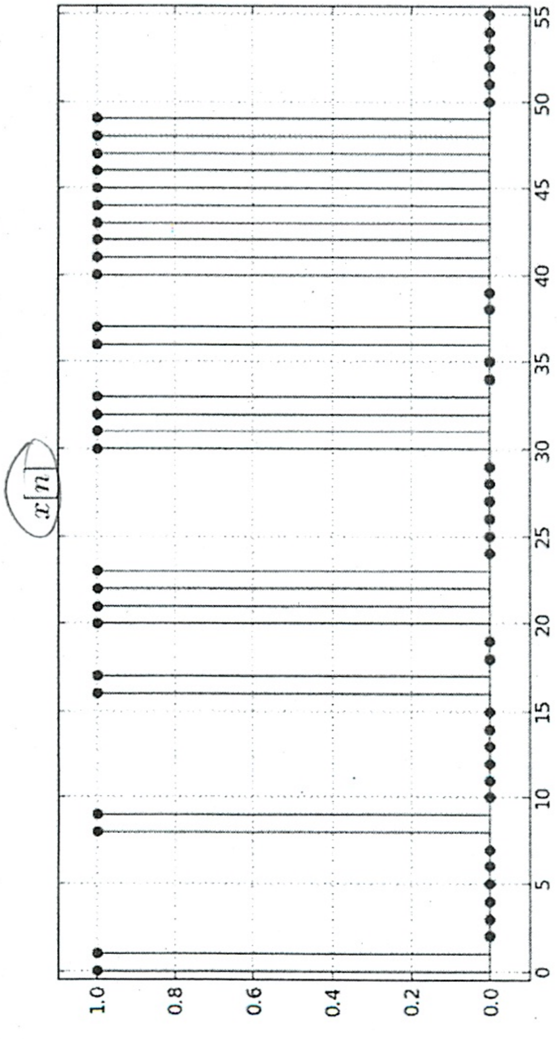
Square since sys · sys

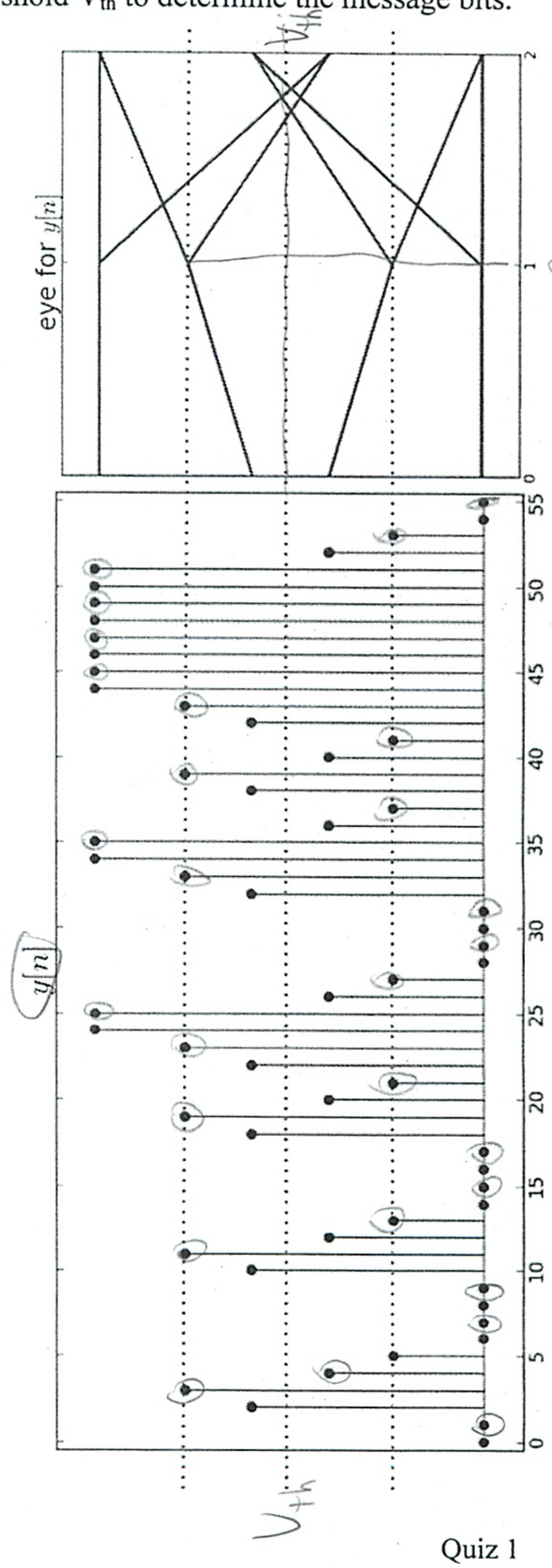
One sys ~~1.8 | 1.8, .5 | 1.8+.7+.5 | .5+.7 | .7~~

same

Consider digital transmissions over the original channel C1 where we use *2 samples/bit*. The following figure shows a test sequence x[n], the channel's response y[n] and an eye diagram constructed from y[n]. Assume x[i] = 0 for i < 0. Note that there are no vertical scales on the plots for y[n] and the eye diagram, but both plots use the same vertical scale (which is *not* the same vertical scale used to plot x[n] – you can't get the answers by measuring!). The receiver will periodically sample y[n] at the widest part of the eye and compare those voltages against a digitization threshold V_{th} to determine the message bits.







? yeah just at middle the

do need to

Find unit sample
$U[n] = s[n] - s[n-1]$

(C) (10 points) What are the possible voltages the receiver will see when it periodically samples y[n] at the widest part of the eye? Since the diagrams have no scale, you will need to compute the voltage values. To receive credit for this part you must *show your work.*

Possible voltage values at sample point: 0, .7, 2.3, 3 ✓

So at time 1 what are all possible voltage values?
Remememember

n	0	1	2	3	4	5 ...
y[n]	0	0	1.8	2.3	3	3 ...

5 Samples involved – but chart only has 2?
– well here since delay, treat as 3

000 = 0
001
010 ← will be key in eye – at 3rd time in so, 0 + 2.3 + 0 = 2.3 which is 76% of total – matches line visually
100
011
101 ← will be key in eye 3 – 2.3 + 0 = .7
110
111 = 3

Walked together

(D) (6 points) Referring to the figure for y[n], give the first three indices for y[n] where the receiver will sample to determine the first 3 bits of the message.

First index: 1 **Second index:** 3 **Third index:** 5 ✗ ⓪

the middle

3, 5, 7 ⇒ the 2 sec delay

(E) (3 points) Assuming there is an equal probability of sending 0's and 1's, what value of V_{th} will maximize the noise margins at the receiver?

right down the middle $\frac{3}{1.5} = 1.5$

Value of V_{th}: 1.5 ✓

(F) (3 points) What is the noise margin in volts using your threshold of part (E)?

? The potential for noise
$2.3 - 1.5 = .8$

Noise margin: .8 ✓

(G) (9 points) Since the C1 channel is noise-free (obviously this a work of fiction), it is possible to reliably use deconvolution to construct a perfect estimate, w[n], of the input waveform given y[n] and $h_{C1}[n]$. Give an equation for w[n] where the only variables are from the response (y[n], y[n-1], y[n+1], …) and earlier values of w (w[n-1], w[n-2], …), everything else must be numeric. In other words, use numeric values for any h_{C1} elements appearing in the equation.

Give equation for w[n]

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

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

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

But cut the leading 0s off so

$h[0] = 1.8$
$h[1] = .5$
$h[2] = .7$
$h[>2] = 0$

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

for the delay →

$$= \frac{y[n+2] - (.5\,w[n-1] + .7\,w[n-2])}{1.8}$$

Should have put example on sheet

8 −4

(H) (5 points) The lecture slides and notes discuss some criteria under which the deconvolution equation will be stable in the presence of noise, i.e., where the estimate w[n] will not grow without bound if some of the y[n] have been affected by noise. Does $h_{C1}[n]$ meet this criteria? Briefly explain.

Brief explanation

Stability if

$$\sum_{m=1}^{k} \left| \frac{h[m]}{h[0]} \right| < 1 \qquad \sum_{m=1}^{k} h[m] < h[0]$$

$.5 + .7 < 1.8$

$1.2 < 1.8$

So system is stable since it meets the stability test. ✓

I thought this was something to apply guessing

END OF QUIZ 1!

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

6.02 Introduction to EECS II
Spring 2011

Quiz 1

Name SOLUTIONS	*Score*

Please write your answers legibly in the spaces provided. You can use the backs of the pages if you need extra room for your answer or scratch work. Make sure we can find your answer!

You can use a calculator and one 8.5" x 11" cribsheet.

Partial credit will only be given in cases where you show your work and (very briefly) explain your approach.

Problem	Score
#1 (30 points)	
#2 (20 points)	
#3 (50 points)	

Problem 1. Information, Entropy and Huffman Codes (30 points)

There's a weekly surprise party at a local independent living group with an equal probability that the event will happen on any of the seven days.

(A) (3 points) You learn that party won't be on the weekend, i.e., not Saturday or Sunday. Give an expression for the number of bits of information you have received.

Expression for number of bits of information received: $\log_2(7/5)$

We've gone from N=7 equally-probable outcomes down to M=5 equally-probable outcomes, so bits of information is $\log_2(N/M)$.

(B) (4 points) Give an expression for the expected length in bits of a Huffman encoding of a message that lists the day of the party for each week of the 52-week year, i.e., a message consisting of 52 variable-length symbols, where each day is encoded separately using the Huffman code. The choice for each week is independent of the choices for other weeks.

Expression for expected length of message in bits: 52*((1/7)*2 + (6/7)*3=52*(20/7)

The Huffman algorithm for 7 equally-probable symbols will build a tree with a depth of 3 for 6 of the symbols and depth of 2 for the seventh symbol.

Examining the historical record, you discover that the probabilities for party days aren't in fact equal – weekends are very popular and the party is never held on Wednesday when 6.02 psets are due. You prepare the following table showing the updated probabilities, which should be used when answering the following questions.

day	*Mon*	*Tue*	*Wed*	*Thu*	*Fri*	*Sat*	*Sun*
p(day)	0.125	0.125	0	0.125	0.125	0.25	0.25
$\log_2(1/p)$	3	3	--	3	3	2	2
$p*\log_2(1/p)$	0.375	0.375	--	0.375	0.375	0.5	0.5
Encoding from part (C)	101	100	--	001	000	11	01

(C) (6 points) Using the updated probabilities, create a variable-length Huffman code for sending messages listing party days. Note that no code is required for Wednesday. Please enter the encoding for each day in the last row of the table above.

Fill in last table row

The Huffman algorithm will build a tree where M,Tu,Th,F have a depth of 3 and Sa, Su have a depth of 2. Any code consistent with these constraints is okay as long as none of the encoding is the prefix of another.

(D) **(4 points) Compute the expected length in bits to encode message containing one day using your code from part (C). Please give a numeric answer.**

Expected length in bits: 2.5

Expected length = Sum of p(sym)*len(encode(sym))
= 0.125*(3+3+3+3) + 0.25*(2+2) = 0.125*12 + 0.25*4

(E) **(4 points) Using the updated probabilities, compute the entropy of the underlying probability distribution. Please give a numeric answer. Hint: much of the computation has already been performed for you!**

Entropy: 2.5

entropy = Sum of p(sym)*log2(1/p(sym)) = 0.375*4 + 0.5*2

(F) **(4 points) By changing the encoding scheme (say, by encoding pairs of days), would it be possible to improve the expected length of messages? Briefly explain why or why not.**

Brief explanation

It's not possible to improve on the expected length of messages by changing the encoding since the expected length of the encoding of part (C) already equals the entropy, which we know is a lower bound on the expected length of messages that deliver the required information.

(G) **(5 points) A phone call from a friend causes you to revise the probabilities for the coming week as follows:**

day	*Mon*	*Tue*	*Wed*	*Thu*	*Fri*	*Sat*	*Sun*
p(day)	0.1	0.1	0	0.1	0.1	0.6	0
$log_2(1/p)$	3.322	3.322	--	3.322	3.322	0.737	--
$p*log_2(1/p)$	0.332	0.332	--	0.332	0.332	0.442	--

How many bits of information did the phone call deliver? Please give a numeric answer.

Bits of information from phone call: 0.73

Entropy before phone call, from part (E) = 2.5 bits
Entropy after phone call = 4*.332 + .442 = 1.77 bits
Information in phone call is given by change in entropy = 2.5 – 1.77

That is the one I was having trouble on
- don't think wording is fair!

stupid question

more like change in information

Problem 2. LZW compression (20 points)

An 8-character message was encoded using the LZW encoder whose pseudo-code is shown below:

```
STRING = get input symbol
WHILE there are still input symbols DO
    SYMBOL = get input symbol
    IF STRING + SYMBOL is in the string table THEN
        STRING = STRING + SYMBOL
    ELSE
        output the code for STRING
        add STRING + SYMBOL to the string table
        STRING = SYMBOL
    END
END
output the code for STRING
```

When the encoding process was complete the following additions had been made to the string table:

table[256] = ho
table[257] = oh
table[258] = hoh
table[259] = hoho

(A) (10 points) What was the original 8-character message?

Original message: hohohoho

Observe from the pseudo-code that additions to the string table are STRING + SYMBOL where the index for STRING is what's sent. So simply by stripping the last character from the table entries we can read off all but the last part of the message: h, o, ho, hoh. From the last entry we know that the last symbol group starts with SYMBOL = o. Since there are no further entries, that means the message ends with either 'o' or 'oh'. We're told that the message is 8 characters, so the message must have been hohohoho.

(B) (10 points) Recall that the encoder only sends indices into the string table. What indices did the encoder send? Hint: everything can be figured out from the string entries and their order. The index of 'h' is 104 and of 'o' is 111.

Indices sent by encoder: 104, 111, 256, 258, 111

This is what gets transmitted encoding the message from part (A) – the transmitter sends the codes for 'h', 'o', 'ho', 'hoh', 'o'

Problem 3. LTI Models for Communication Channels (50 points)

Consider a communications channel *C1* that is accurately modeled as a noise-free linear time invariant system with the following causal unit sample response:

$h_{C1}[0]$	$h_{C1}[1]$	$h_{C1}[2]$	$h_{C1}[3]$	$h_{C1}[4]$	$h_{C1}[\geq 5]$
0.0	0.0	1.8	0.5	0.7	0.0

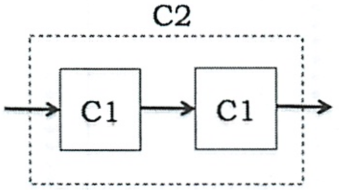
(A) (4 points) The unit step response for this channel, $s_{C1}[n]$, eventually reaches a steady state value v. What is v and what is the smallest k such that $s_{C1}[k] = v$?

Steady state value v: 3.0

Smallest k: 4

s[n] = u[n]*h[n] = [0, 0, 1.8, 2.3, 3.0, 3.0, 3.0, …]

(B) (10 points) Suppose we built a communications channel *C2* composed of two C1 channels connected in series:



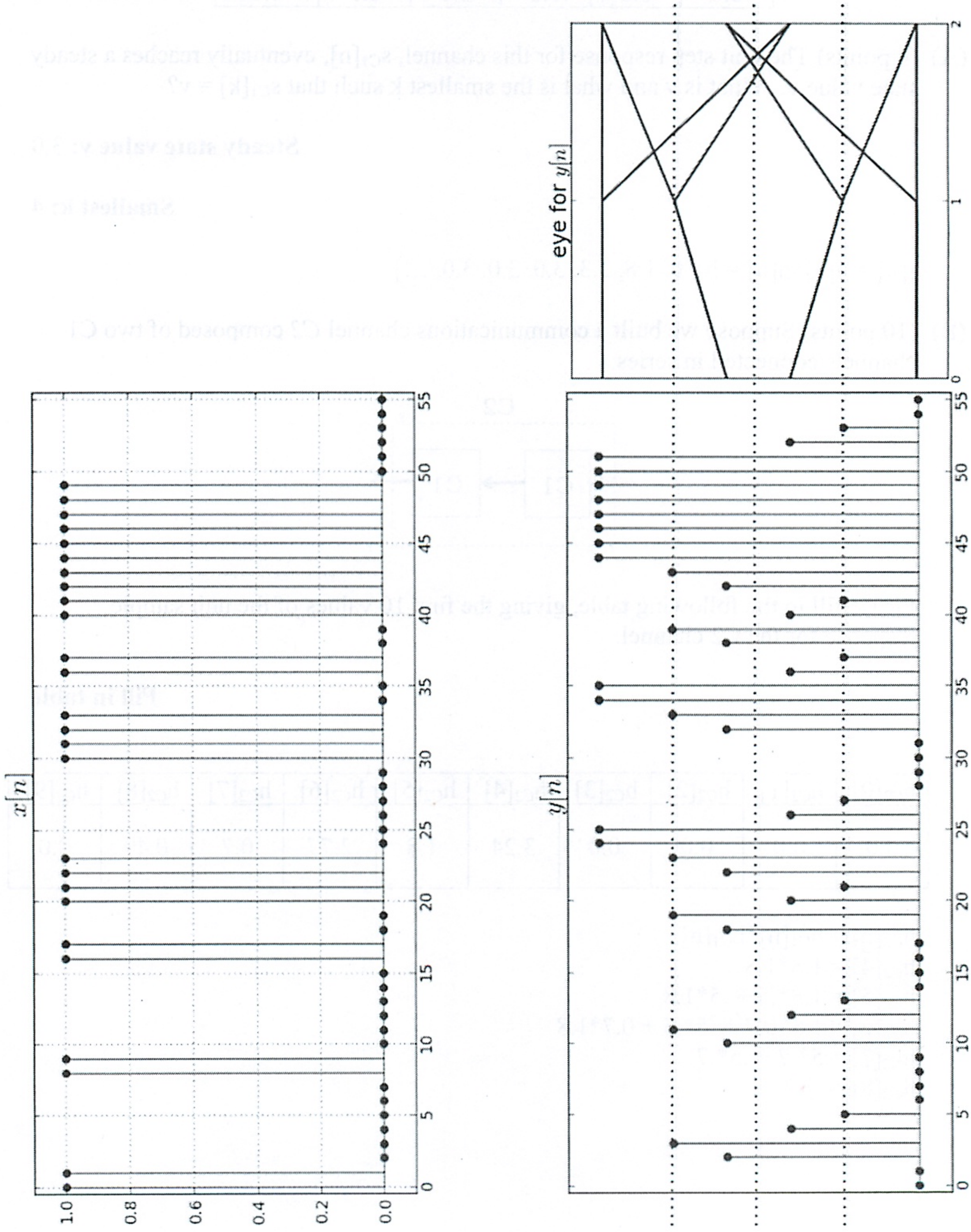
Please fill in the following table, giving the first 10 values of the unit sample response for the C2 channel.

Fill in table

$h_{C2}[0]$	$h_{C2}[1]$	$h_{C2}[2]$	$h_{C2}[3]$	$h_{C2}[4]$	$h_{C2}[5]$	$h_{C2}[6]$	$h_{C2}[7]$	$h_{C2}[8]$	$h_{C2}[9]$
0.0	0.0	0.0	0.0	3.24	1.8	2.77	0.7	0.49	0.0

$h_{C2}[n] = h_{C1}[n]*h_{C1}[n]$
$h_{C2}[4] = 1.8*1.8$
$h_{C2}[5] = 1.8*.5 + .5*1.8$
$h_{C2}[6] = 1.8*0.7 + .5*.5 + 0.7*1.8$
$h_{C2}[7] =.5*.7 + .5*.7$
$h_{C2}[8] = .7.7$

Consider digital transmissions over the original channel C1 where we use *2 samples/bit*. The following figure shows a test sequence x[n], the channel's response y[n] and an eye diagram constructed from y[n]. Assume x[i] = 0 for i < 0. Note that there are no vertical scales on the plots for y[n] and the eye diagram, but both plots use the same vertical scale (which is *not* the same vertical scale used to plot x[n] – you can't get the answers by measuring!). The receiver will periodically sample y[n] at the widest part of the eye and compare those voltages against a digitization threshold V_{th} to determine the message bits.



(C) (10 points) What are the possible voltages the receiver will see when it periodically samples y[n] at the widest part of the eye? Since the diagrams have no scale, you will need to compute the voltage values. To receive credit for this part you must *show your work.*

Possible voltage values at sample point: 0.0, 0.7, 2.3, 3.0

Use convolution sum to compute y[k] where y[k] = voltage in eye diagram (avoid y[0] and y[1] since they are due to 2-sample delay in channel)

lowest voltage (k=6): y[6] = 0*x[6] + 0*x[5] + 1.8*x[4] + .5*x[3] + .7*x[2] = 0.0

next voltage (k=5): y[5] = 0*x[5] + 0*x[4] + 1.8*x[3] + .5*x[2] + .7*x[1] = 0.7

next voltage (k=11): y[11] = 0*x[11] + 0*x[10] + 1.8*x[9] + .5*x[8] + .7*x[7] = 2.3

highest voltage (k=24): y[24] = 0*x[24] + 0*x[23] + 1.8*x[22] + .5*x[21] + .7*x[20] = 3.0

(D) (6 points) Referring to the figure for y[n], give the first three indices for y[n] where the receiver will sample to determine the first 3 bits of the message.

First index: 3 **Second index:** 5 **Third index:** 7

Sample at the widest part of the eye, taking into account 2-sample delay.

(E) (3 points) Assuming there is an equal probability of sending 0's and 1's, what value of V_{th} will maximize the noise margins at the receiver?

Value of V_{th}: 1.5

Maximize noise margin by choosing voltage a mid-point of eye.

(F) (3 points) What is the noise margin in volts using your threshold of part (E)?

Noise margin: 2.3-1.5 = 0.8

(G) (9 points) Since the C1 channel is noise-free (obviously this a work of fiction), it is possible to reliably use deconvolution to construct a perfect estimate, w[n], of the input waveform given y[n] and $h_{C1}[n]$. Give an equation for w[n] where the only variables are from the response (y[n], y[n-1], y[n+1], …) and earlier values of w (w[n-1], w[n-2], …), everything else must be numeric. In other words, use numeric values for any h_{C1} elements appearing in the equation.

Give equation for w[n]

w[n] = (1/1.8) * (y[n+2] - .5*w[n-1] - .7*w[n-2])

To eliminate channel delay and ensure a non-zero h[0], we need to shift h[n] and y[n] by 2 to the left, which we can accomplish by adding 2 to their indices in the standard deconvolution equation.

(H) (5 points) The lecture slides and notes discuss some criteria under which the deconvolution equation will be stable in the presence of noise, i.e., where the estimate w[n] will not grow without bound if some of the y[n] have been affected by noise. Does $h_{C1}[n]$ meet this criteria? Briefly explain.

Brief explanation

The notes say the deconvolution will be stable if Σ abs(h[m])/abs(h[0]) < 1.

.5/1.8 + .7/1.8 = 1.2/1.8 < 1. So $h_{C1}[n]$ meets this criterion.

END OF QUIZ 1!

0.0 0.7 0.5 1.8 0.0 0.0

0.0 0.0 0.0 0.0 1.8^2 1.8

$2(1.8 * .7)$
$+ .25$

ρ δ

[2] L = 1.8

[3] .5

[4] = .7

-k

	[illegible]	v
	n = 10	.43
	= 20	.26
	= 40	.16
	= 80	.09

.25	.4
.2	.4
.3	.36
.4	.12
.35	.28
.15	.38

0.0 0.0 1.2 0.5 0.7 0.0

	0	1	2	3	4
Sys 1 out = Sys 2 in			1.8	.5	.7

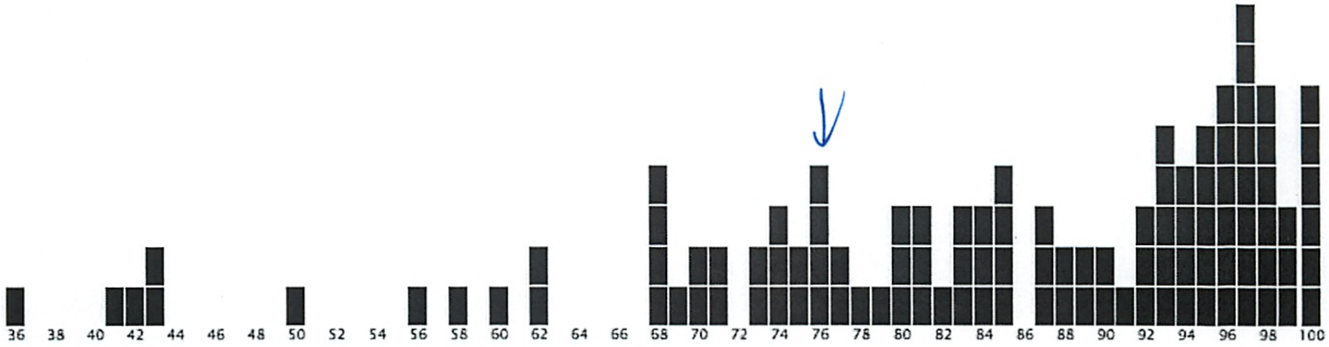
6 Uncertanity ellіminate

reduces uncetanity in day

info = Δ entropy

lot of probs go to 0

Histogram for Q1



¿How did I do so poorly? I thought I understand it

Its all these issues that add up

6.02 #1

Information - ↓ uncertanity

Info content $\log_2 \frac{1}{P(seq)}$ in bits

Expected info content

$$H(X) = E(I(X)) = \sum_{i=1}^{n} p(x_i)\log_2\left(\frac{1}{p(x_i)}\right)$$

If all p_i's =, uniform, $\frac{1}{N}$

$\log_2(N)$

Outcome reduced to M possible choices

Entropy after reciept

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

Entropy in <u>message</u> is change

$H_{before} - H_{after} = \log_2(N) - \log_2(M)$

$= \log_2\left(\frac{N}{M}\right)$ original # choices

↑ # of choices left!

Example: Have 52 cards, tell you its a ♢

So $\log_2\left(\frac{52}{13}\right) = 2$ bits info

↑ I gave you assuming all prob =

Or add $\log_2\left(\frac{1}{.2+.2+.2}\right)$

↑ remaining probs

Additive

Fixed is eaiset

Variable saves space

- max down to entropy

Huffman coding - optimal

Compute avg lenght of code

$\sum P_{symbol}(L_{sympol})$

$p \sim \frac{1}{2^k}$

Log the power to which the base must be raised to do produce that #

$\log_2 16 \rightarrow 2^x = 16$

$\log(xy) = \log(x) + \log(y)$

$\log_b(x^p) = p\log_b x$

$e^{\ln(x)} = x \quad \ln(e^x) = x$

$\log_b\left(\frac{x}{y}\right) = \log_b(x) - \log_b(y)$

$\log_b(\sqrt[p]{x}) = \frac{\log_b(x)}{p}$

$\log_b(x) = \log_k(x)/\log_k(b)$

Clock Recovery The constant adjust forward/backwards

- Sample middle one

Other stuff

How many samples/bit

Where does byte start?

8b/10

1. Lots of bit transitions
2. DC balance 0s, 1s
3. Special sync symbol

$x[n]$ = input

$y(n)$ = output

$u[n]$ = unit step

$s[n]$ = unit step response

$\delta[n]$ = unit sample ⊥

$h[n]$ = unit sample response/chanel description fading coefficent

break everything down to unit samples

Time invaried $x[n-N] \rightarrow y[n-N]$

Linear $ax_1[n] + bx_2[n] \rightarrow ay_1[n] + by_2[n]$

weighted sums in → out

allows: deconvolution

Convolution

- commutative $x[n]*h[n] = h[n]*x[n]$
- associative
  $x[n]*(h_1[n]*h_2[n]) = (x[n]*h_1[n])*h_2[n]$
- distributive
  $x[n]*(h_1[n]+h_2[n]) = x[n]*h_1[n] + x[n]*h_2[n]$

Parallel

Series

Just a product of what came before

Just clearest point - not when transition is

Causal - depends only on current + previous values

Scalar - real #, not vector

$u[n] = s[n] - s[n-1]$

ISI

$$B = \left\lceil\frac{\text{lenght } h[n] \text{ active}}{N}\right\rceil + 2$$

test pattern $2^N \cdot B$

pick samples/bit

diff than fast/slow channel

deconvoltion

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

Stability

$$\sum_{m=1}^{k}\left|\frac{h[m]}{h[0]}\right| < 1 \quad \sum_{m=1}^{k} h[m] < h[0]$$

drop first $h[0]$ if is or close to 0

Noise

Mean $\mu_x = \frac{1}{N}\sum_{n=1}^{N} x[n]$

$$P_x = \frac{1}{N}\sum_{n=1}^{N} x[n]^2 \quad \hat{P}_x = \frac{1}{N}\sum_{n=1}^{N}(x[n]-\mu_x)^2$$

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

$$SNR = \frac{\hat{P}_{signal}}{\hat{P}_{Noise}} \quad SNR(db) = 10\log\left(\frac{P_{signal}}{P_{noise}}\right)$$

stationary vs ergodic random processes

$$P(x_1 \le X \le x_2) = \int_{x_1}^{x_2} f_X(x)\,dx$$

$$\mu_X = \int_{-\infty}^{\infty} x f_X(x)\,dx$$

$$\sigma^2 = \int_{-\infty}^{\infty}(x-\mu_X)^2 f_X(x)\,dx$$

$$PDF_{normal} = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\left(\frac{-(x-\mu)^2}{2\sigma^2}\right)}$$

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

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

BER bit error ratio

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

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

$$= \frac{1}{N}\sum_{n=1}^{N}\left(\frac{1}{2}\right)^2 = \frac{1}{N}\frac{N}{4} = \frac{1}{4}$$

$$P(\text{error}) = P(0)\cdot P(\text{error}|\text{trans }0) + P(1)P(\text{error}|1)$$

$$= .5\cdot\Phi(-.5/\sigma) + .5\cdot\Phi(-.5/\sigma)$$

$$= \Phi(-.5/\sigma)$$

$$SNR(db) = 10\log\left(\frac{\tilde{P}_{signal}}{\tilde{P}_{noise}}\right) = 10\log\left(\frac{.25}{\sigma^2}\right)$$

Eye diagram

all possible voltage sequences in a certain # of bits

If have a channel that reciens more 1s make it more likely to reciee a 1

Find Φ w/ test signals

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

Deconvolution   w is estimated x

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

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

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

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

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

Encode LZW

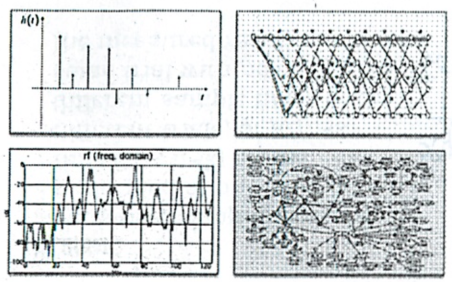
```
initalize TABLE[0 to 255] = code for
↳ individual bytes
STRING = get input signal
while there are still input symbols:
    SYMBOL = get input symbol
    if STRING + SYMBOL is in TABLE:
        STRING = STRING + SYMBOL
    else:
        output the code for STRING
        add STRING + SYMBOL to
                          ↳ TABLE
        STRING = SYMBOL
output the code for STRING
```

Decode LZW

```
initalize TABLE[0 to 255] = code for
↳ individual bytes
CODE = read next code from encoder
STRING = TABLE[CODE]
output STRING

while there are still codes to recieve:
    CODE = read next code from encoder
    if TABLE[CODE] is not defined:
        ENTRY = STRING + STRING[0]
    else:
        ENTRY = TABLE[CODE]
    output ENTRY
    add STRING + ENTRY[0] to TABLE
    STRING = ENTRY
```

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

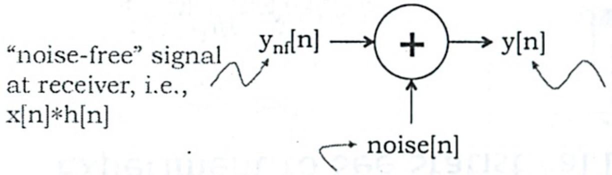
Noise, broadly construed, is any change to the signal from its expected value, x[n]*h[n], when it arrives at the receiver.

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]

noise[n]

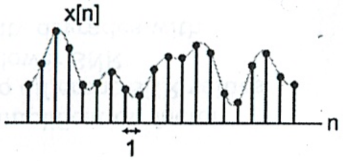
"noisy" signal receiver must process

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

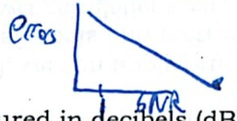
Energy: $E_x = \sum_{n=1}^{N} x[n]^2$ $\tilde{E}_x = \sum_{n=1}^{N}(x[n]-\mu_x)^2$

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

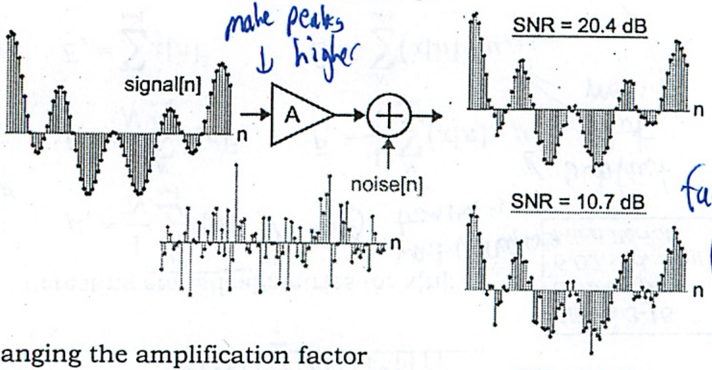
3db is a factor of 2

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

2/22

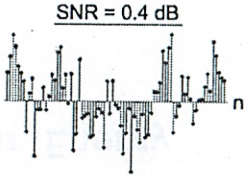
SNR Example

make peaks ↓ higher



factor of 10

10 x more signal than noise



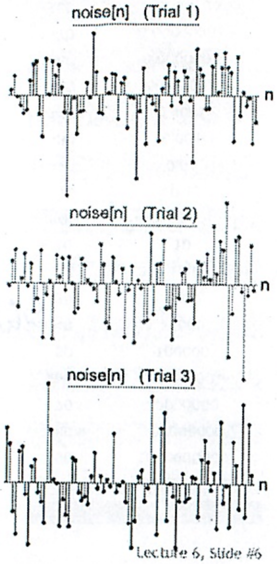
near the same

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

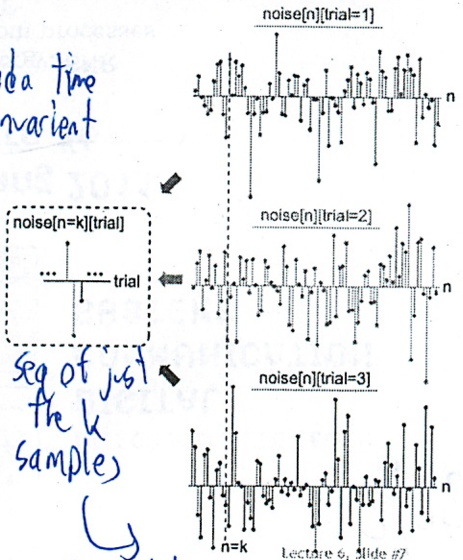
Stationary

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

kinda time invariant

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

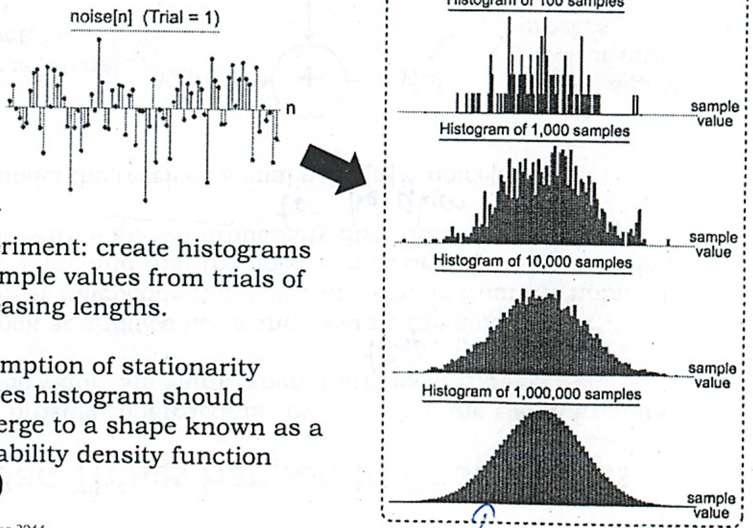


seq of just the k samples

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)

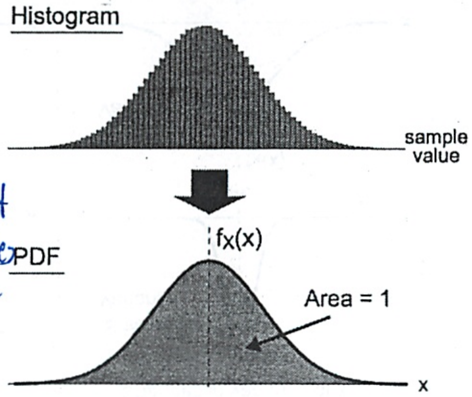
Converges to Normal

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



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

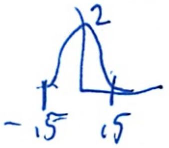
area must = 1

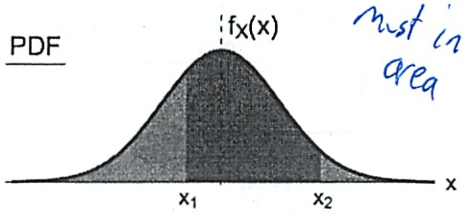
Formalizing Probability

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

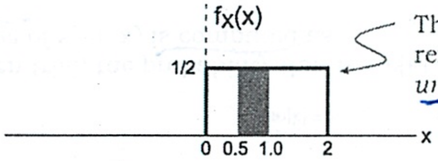




must integrate area

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:

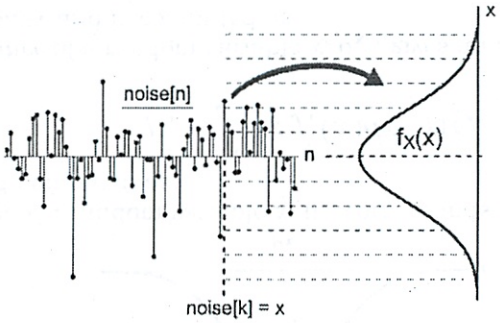
★ must integrate

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

?

Can't just read #

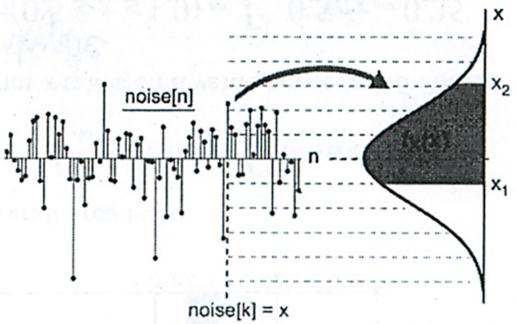
Examination of Sample Value Distribution



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*.

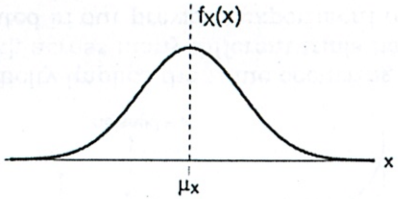
Probability Calculation



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:

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

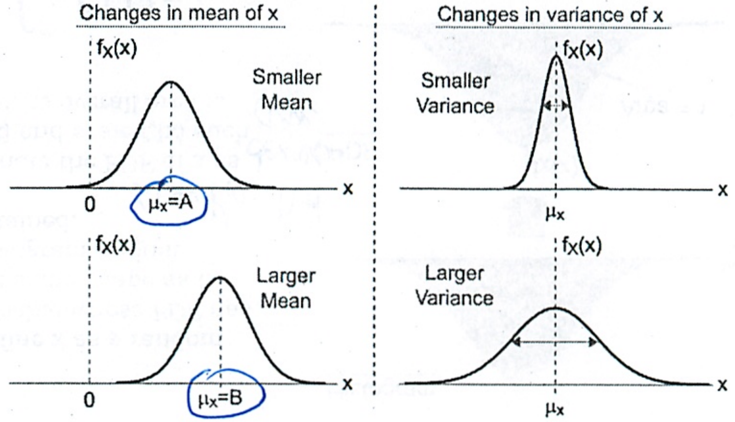
Mean shall be 12 for noise

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

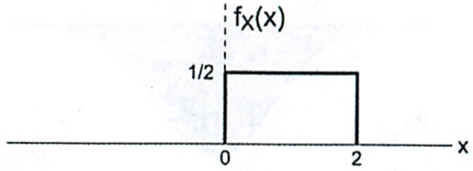
Visualizing Mean and Variance



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:

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

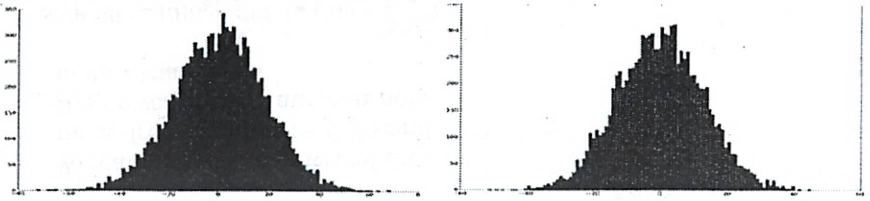
Variance:

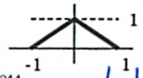
$$\sigma_x^2 = \int_{-\infty}^{\infty} (x-\mu_x)^2 f_x(x)\,dx = \int_0^2 (x-1)^2\frac{1}{2}dx = \frac{1}{6}(x-1)^3\Big|_0^2 = \frac{1}{3}$$

Noise on a Communication Channel

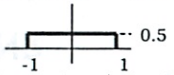
The net noise observed at the receiver is often the sum of many small, independent random contributions from the electronics and transmission medium. If these independent random variables have finite mean and variance, the Central Limit Theorem says their sum will be *normally* distributed.

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.





Triangular PDF



Uniform PDF

but still tends to be normal distribute d

better in color

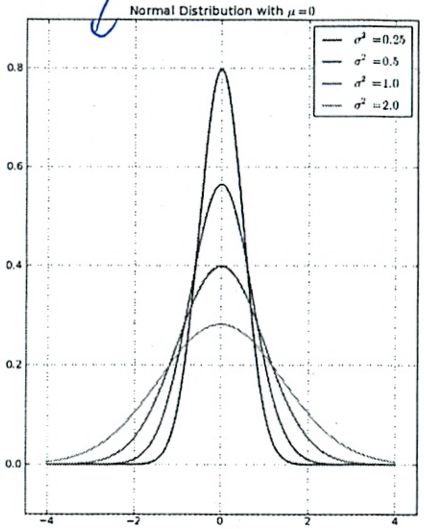
The Normal Distribution

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.



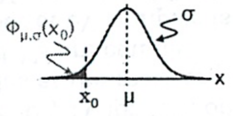
all 0 mean

Cumulative Distribution Function ϕ

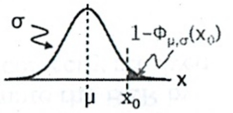
When analyzing the effects of Gaussian noise, we'll often want to determine the probability that the noise is larger or smaller than a given value x_0. From slide #10:

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).

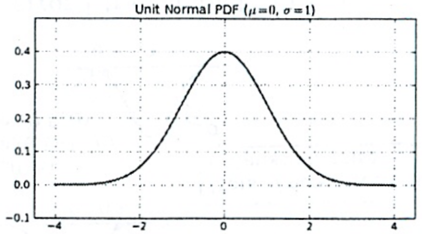
$$\boxed{\Phi_{\mu,\sigma}(x)=\Phi\left(\frac{x-\mu}{\sigma}\right)}$$

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*:

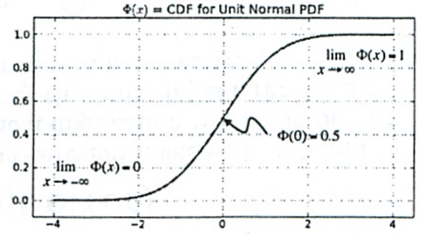
$$\operatorname{erf}(x)=\frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$$



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

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

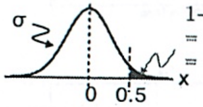
$$\tilde{P}_{y_{nf}} = \frac{1}{N}\sum_{n=1}^{N}\left(y_{nf}[n]-\frac{1}{2}\right)^2 = \frac{1}{N}\sum_{n=1}^{N}\left(\frac{1}{2}\right)^2 = \frac{1}{N}\frac{N}{4} = \frac{1}{4}$$

use prob to predict error rate

p(bit error)

Now assume the channel has Gaussian noise with μ=0 and variance σ². 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"):

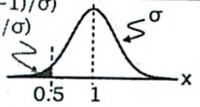


$1-\Phi_{\mu,\sigma}(0.5) = \Phi_{\mu,\sigma}(-0.5)$
$= \Phi((-0.5-0)/\sigma)$
$= \Phi(-0.5/\sigma)$

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)

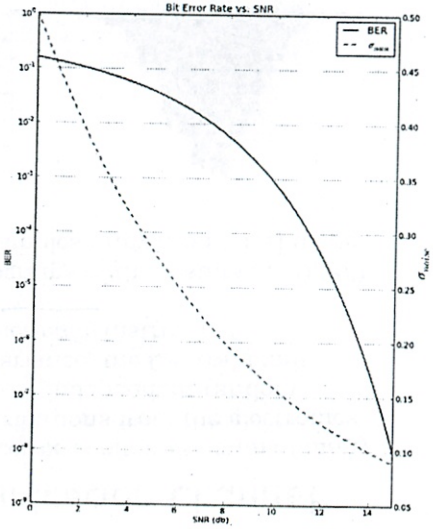
BER (no ISI) vs. SNR

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

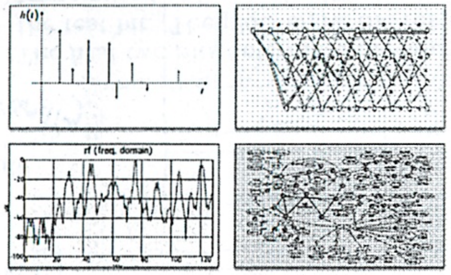
$$\text{SNR (db)} = 10\log\left(\frac{\tilde{P}_{signal}}{\tilde{P}_{noise}}\right) = 10\log\left(\frac{0.25}{\sigma^2}\right)$$

Given an SNR, we can use the formula above to compute σ² 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

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

$\uparrow$ noise[n] not exact, but can get some info → gaussian

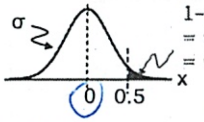
Var $= \sigma^2$

p(bit error)

Now assume the channel has Gaussian noise with μ=0 and variance σ². 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:

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

p(error | transmitted "0"):



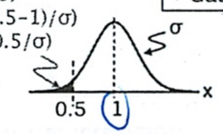
$1-\Phi_{\mu,\sigma}(0.5) = \Phi_{\mu,\sigma}(-0.5)$
$= \Phi((-0.5-0)/\sigma)$
$= \Phi(-0.5/\sigma)$

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)

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

Have some value of interest

Tells us → P(noise[k] ≤ x_0)

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

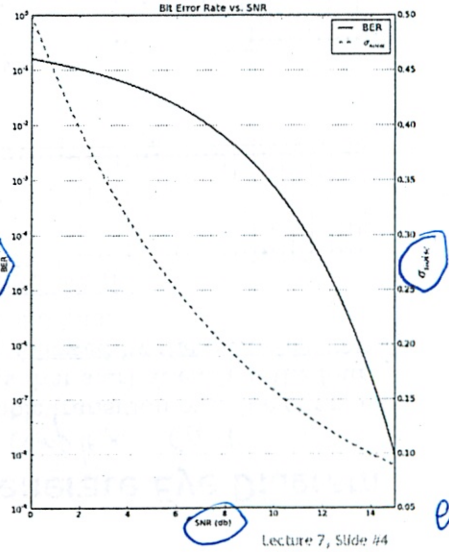
BER (no ISI) vs. SNR

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 σ² 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

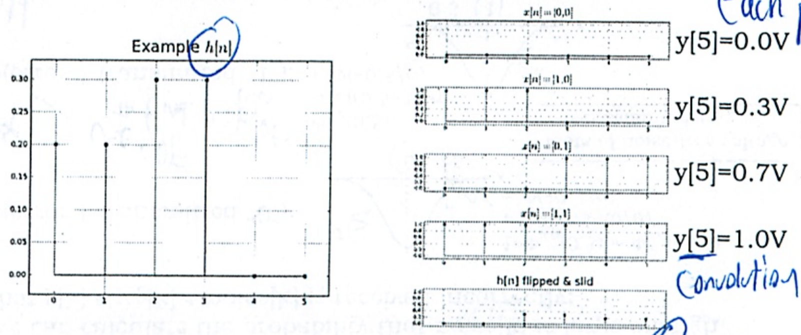
Symbols not perfect 0s or 1s

(What I had questions on)

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

Convolution

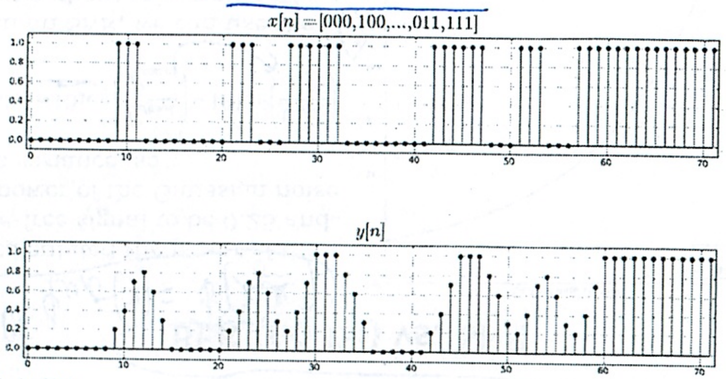
Each possible combo of 2 bits

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

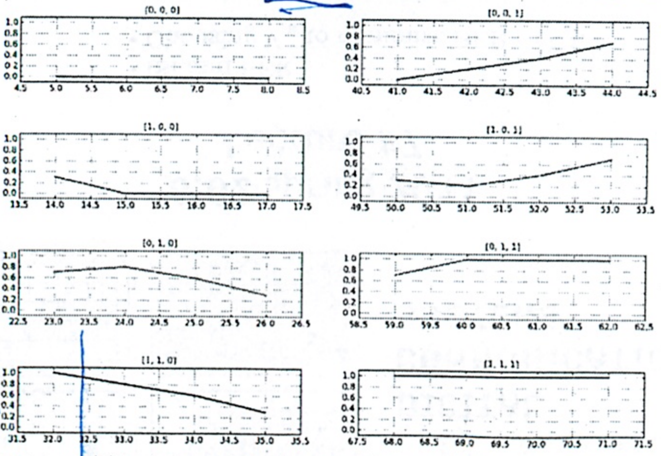
So a more complex case

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 though 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 energe left from previous 1 bit

stick on top of each other

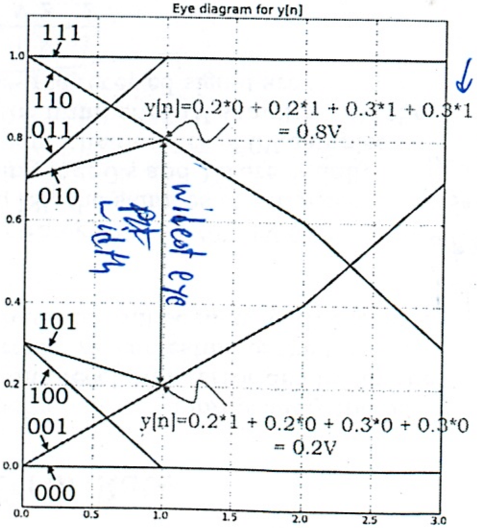
Plot the Eye Diagram

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.

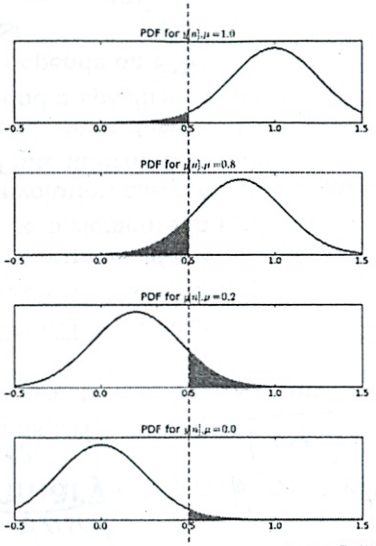
p(bit error) with ISI

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.

p(bit error) = p(11)*p(error | 11) + p(10)*p(error | 10) +
p(01)*p(error | 01) + p(00)*p(error | 00)
= $0.25\cdot\Phi(-0.5/\sigma) + 0.25\cdot\Phi(-0.3/\sigma) +$
$0.25\cdot\Phi(-0.3/\sigma) + 0.25\cdot\Phi(-0.5/\sigma)$
= $0.5\cdot\Phi(-0.5/\sigma) + 0.5\cdot\Phi(-0.3/\sigma)$

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) = $0.5\cdot\Phi(-2) + 0.5\cdot\Phi(-1.2) = 0.069$ 7% - crummy channel

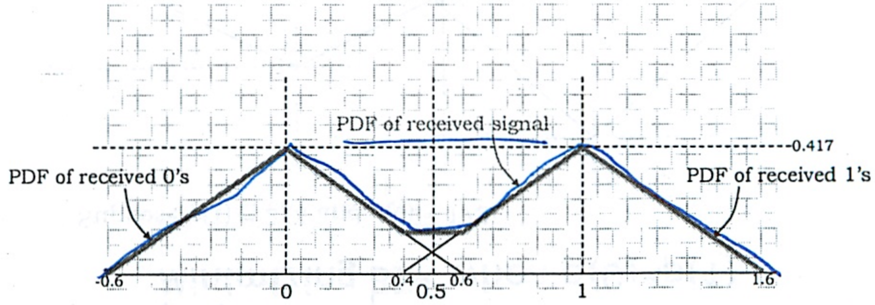
Bottom line: a factor of 3 increase in BER 3x worse

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:

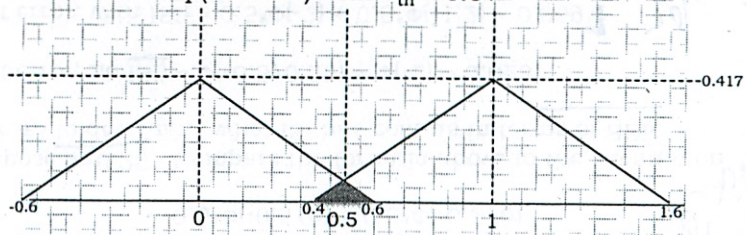


Aggressivly moving through the formulas

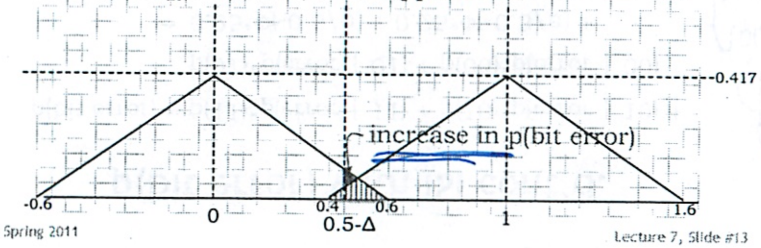
Equal Prob

Minimizing BER

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



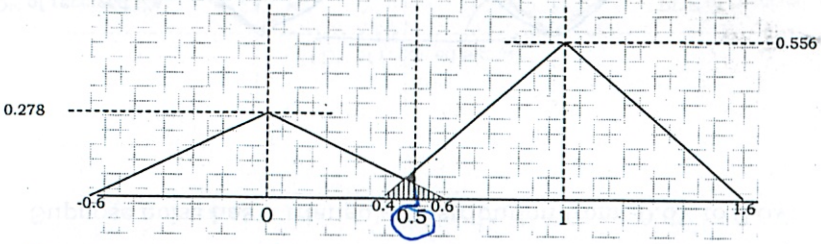
Now move V_{th} slightly. What happens to BER?



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?

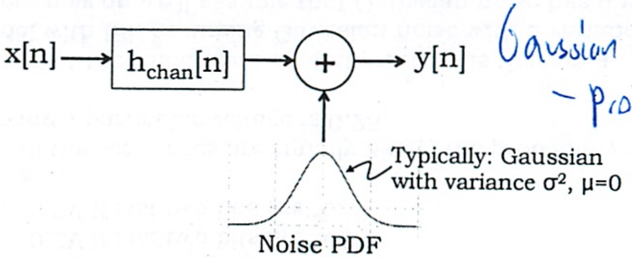
if send more 1s, build a reciever that is better at recieving 1s

put lines where P lines intersect

Could move this triangle to 2 And have 0 error

Gaussian is never cut off - Prob just gets smaller

Channel Model Summary



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

+ Study Guide

— could do math to find it

- 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

Part 2 lecture fairly straightforward

6.02 Recitation

2/24

Probability

$\Omega = \{0,1\}$ ← 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

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

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

Rules

1. $P(\mathcal{E}) \geq 0, \leq 1$

2. $P(\Omega) = 1$

3. $\mathcal{E}_1, \mathcal{E}_2 \quad \mathcal{E}_1 \cap \mathcal{E}_2 = \emptyset$

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

② Gaussian/Normal

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

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

$$= \Phi(a)$$

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

$\quad \shortparallel$

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

$Y = \sigma Z$

$Z \sim N(0,1)$

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

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

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

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

③

$Y = X + \mu$

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

$Y = \sigma Z$

$var(X) = \sigma^2 var(Z)$

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

Take away message

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

$X = \sigma Z + \mu$

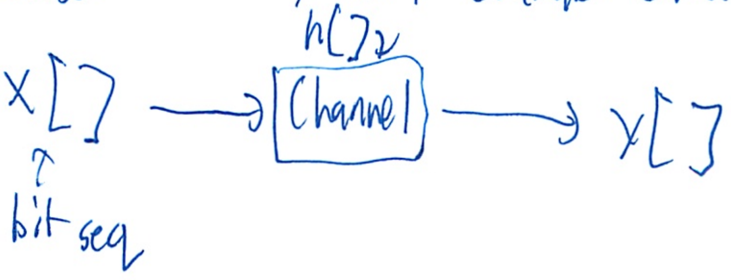
$Z \sim N(0,1)$

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

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

Bit error rate

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



But what if noise is added?

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

Deconvolling is screwed vp by noise

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

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

Deconvolution in a linear opeation

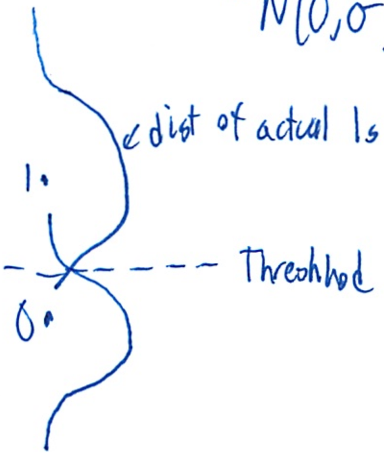
Think of noise as [illegible] guassian dist

Use that to do deconvolve

Tutorial Poblem

Suppose noise added to 0 or 1

$\uparrow N(0,\sigma)$



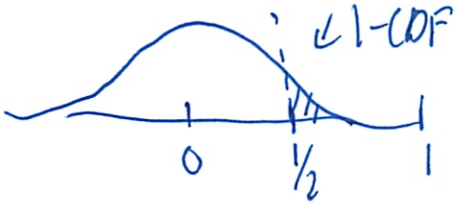
Recieve = In + Noise

(5)

What are the chances stuff screws up

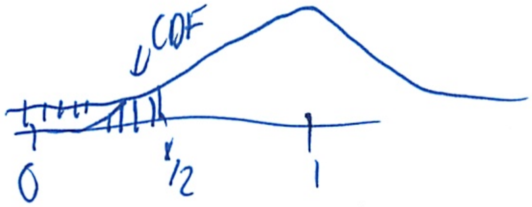
$P_{0\to1}$ $0 \to 1$

sent   recieved



$P_{1\to0}$ $1 \to 0$

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

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

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

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

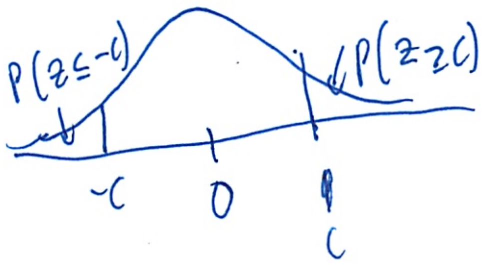
$$= P\left(Z > \frac{V}{2\sigma_{Noise}}\right)$$

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

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

6

$z \sim N(0,1)$



$P(z \geq c) = P(z \leq c)$

Then what is $-z$? Same

So do $1-\Phi(c)$

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

$$\begin{aligned} P_{1\to 0} &= P\left(Rec < \tfrac{V}{2} \,\middle|\, In = V\right) \\ &= P\left(Noise < -\tfrac{V}{2}\right) \\ &= P\left(\sigma_{Noise}\, z < -\tfrac{V}{2}\right) \\ &= P\left(z < \frac{V}{-2\sigma_{Noise}}\right) \\ &= 1-\Phi\left(\frac{V}{2\sigma_{Noise}}\right) \end{aligned}$$

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 ~~set~~ sent at

↳ the difference from 0 which is sent at 0

To calculate σ

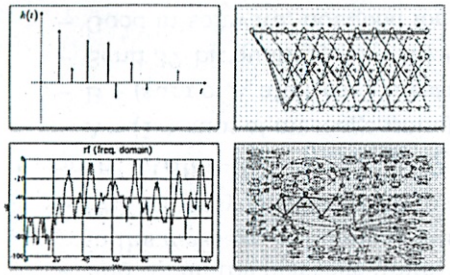
- ~~Set~~ send test signals
- try to measure how much signal has changed

Its relative SNR ratio that matters

If > 0, can transmit something

But not very efficiently

2/28



INTRODUCTION TO EECS II

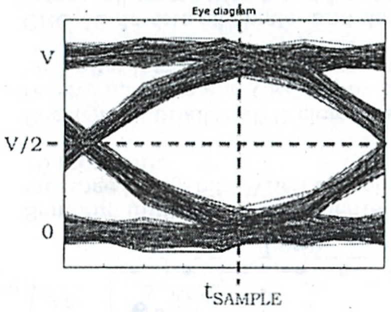
DIGITAL COMMUNICATION SYSTEMS

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

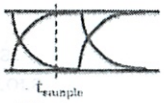
There's good news and bad news...



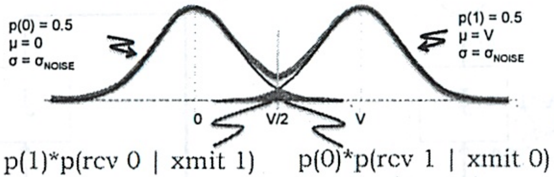
The good news: Our digital signaling scheme usually allows us to recover the original signal despite small amplitude errors introduced by inter-symbol interference and noise. An example of the digital abstraction doing its job!

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:

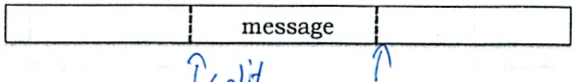
$$BER = \overbrace{(0.5)\Phi\left[\frac{V/2 - V}{\sigma_{NOISE}}\right]} + \overbrace{(0.5)\left[1 - \Phi\left[\frac{V/2 - 0}{\sigma_{NOISE}}\right]\right]} = \Phi\left[\frac{-V/2}{\sigma_{NOISE}}\right]$$

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

transmit 1 got 0

transmit 0 got 1

Dealing With Errors: Packets

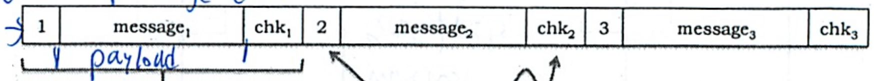


split

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

unique id

package chunks



Packet = {#, message, chk}

Sequence number provides unique identifier for each packet.

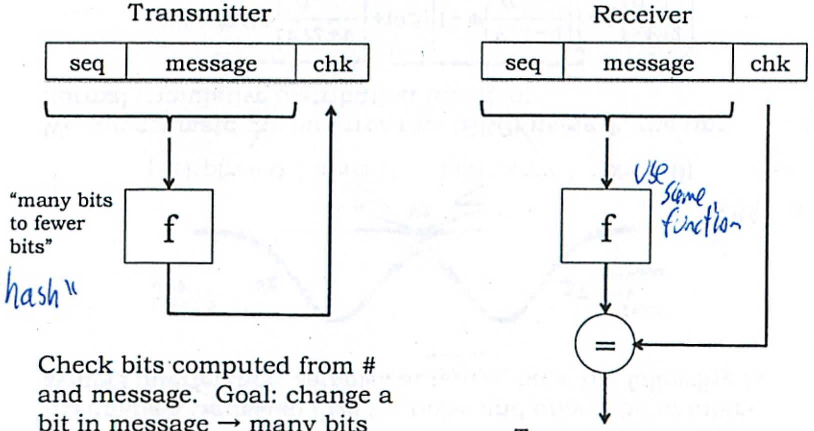
Check bits are redundant information that lets receiver verify # and message. Failure? Ask for packet to be resent.

that don't know # – ~~[illegible]~~ first 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



Transmitter

seq | message | chk

"many bits to fewer bits"

f

"hash"

Receiver

seq | message | chk

f

use same function

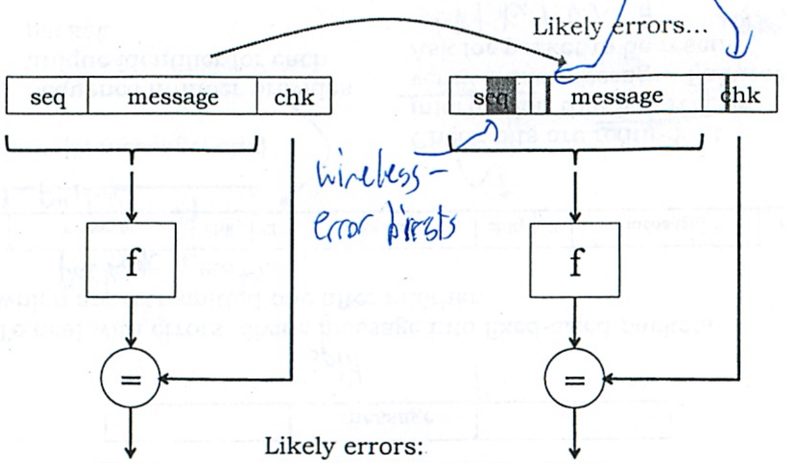
=

True: no errors
False: errors

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...

seq | message | chk

f

=

True

seq | message | chk

f

=

False

wireless - error bursts

Likely errors:
- Random bits (BER)
- Error bursts

err on side of calling good packets bad

Checksums

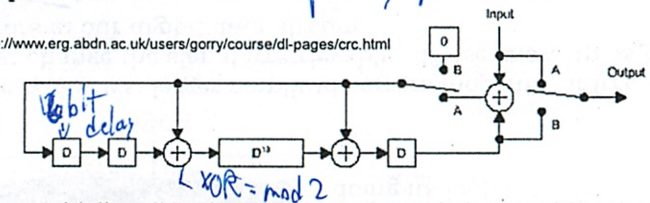
- Simple checksum
  - Add up all the message units, send along sum
  - Easy for two errors to mask one another offset each other
    - Some 0 bit changed to a 1; 1 bit in same position in another message unit changed to a 0... sum is unchanged
- Weighted checksum
  - Add up all the message units, each 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

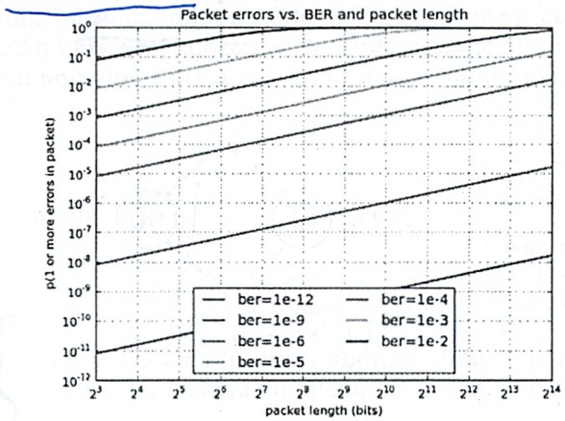
Channel type	Bandwidth	BER
Telephone Landline	2 Mbits/sec	10^{-4} to 10^{-6}
Twisted pair (differential)	1 Gbits/sec	$\leq 10^{-7}$ Ethernet
Coaxial cable	100 Mbits/sec	$\leq 10^{-6}$
Fiber Optics	10 Tbits/sec	$\leq 10^{-9}$
Infrared	2 Mbits/sec	10^{-4} to 10^{-6}
3G cellular	1 Mbits/sec	10^{-4}

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

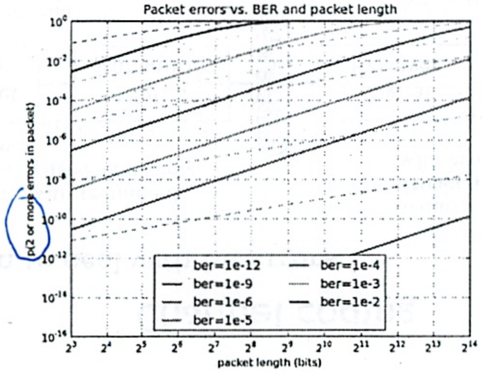


With 1kbyte packets and BER=1e-6, retransmit 1 every 100.

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

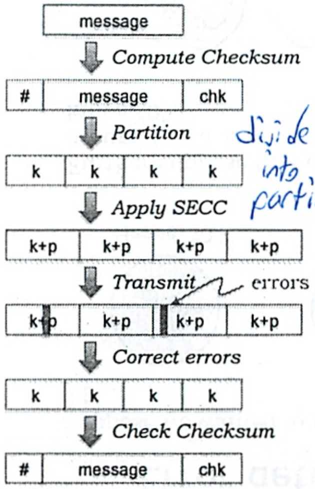


Use this often for hard drives

ECC - error correct.

Use p bit to correct

Digital Transmission using SECC



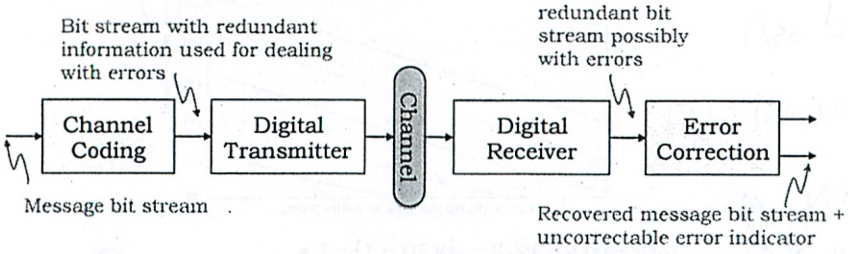
divide into partitions

- 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(√k)
  - Hamming: p grows as O(log k)
- After xmit, correct errors
- Verify checksum, fails if undetected/uncorrectable error
- Deliver or discard message

In troduce end ~~k~~ 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

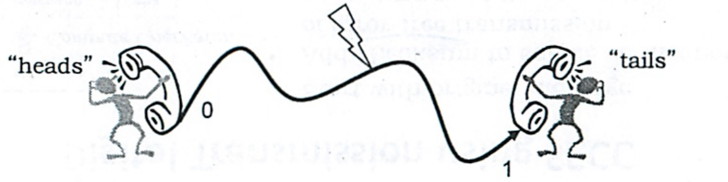
We'll add redundant information to the transmitted bit stream (a process called channel coding) so that we can detect errors at the receiver. Ideally we'd like to correct commonly occurring errors, e.g., error bursts of bounded length. Otherwise, we should detect uncorrectable errors and use, say, retransmission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

Error detection and correction

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

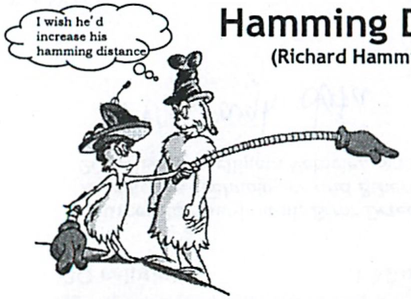
This is a prototype of the "bit" coin for the new information economy. Value = 12.5¢

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

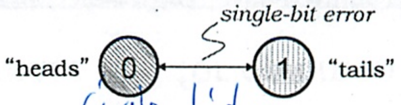
(Richard Hamming, 1950)



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...



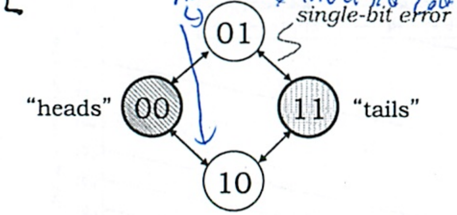
Single bit error

Conv~~[illegible]~~ one value another

Just

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 - screwe!

Parity check

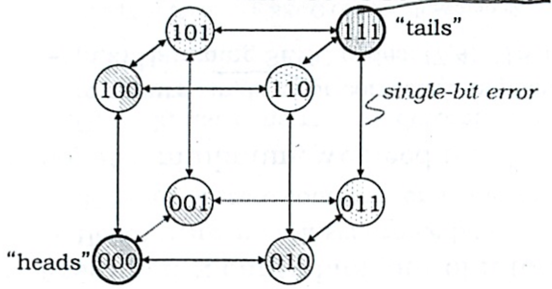
- A parity bit can be added to any length message and is chosen to make the total number of "1" bits even (aka "even parity").
- To check for a single-bit error (actually any odd number of errors), count the number of "1"s in the received message and if it's odd, there's been an error.

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).

Now also

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?

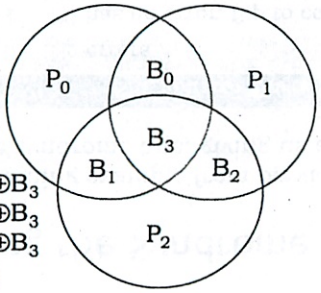
Single Error Correcting Codes (SECC)

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 <u>single error</u> 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

$$E_0 = B_0 \oplus B_1 \oplus B_3 \oplus P_0$$
$$E_1 = B_0 \oplus B_2 \oplus B_3 \oplus P_1$$
$$E_2 = B_1 \oplus B_2 \oplus B_3 \oplus P_2$$

So for each combo. Some is missing ~~missing~~ -so flip that one

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

(did not pay close attention last 30 min)

Using the Syndrome to Correct Errors

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

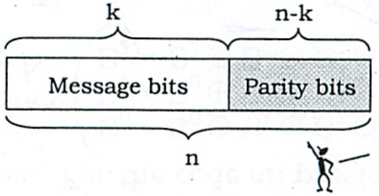
$E_2E_1E_0$	Single Error Correction
0 0 0	No errors
0 0 1	P0 has an error, flip to correct
0 1 0	P1 has an error, flip to correct
0 1 1	B0 has an error, flip to correct
1 0 0	P2 has an error, flip to correct
1 0 1	B1 has an error, flip to correct
1 1 0	B2 has an error, flip to correct
1 1 1	B3 has an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



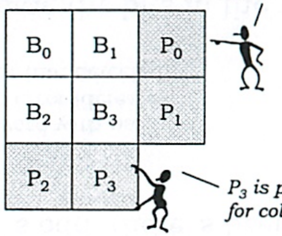
The entire block is called a "code word" and this is an (n,k) code.

- Often we'll use the notation (n,k,d) where d is the minimum Hamming distance between code words. from the def. of the 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



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

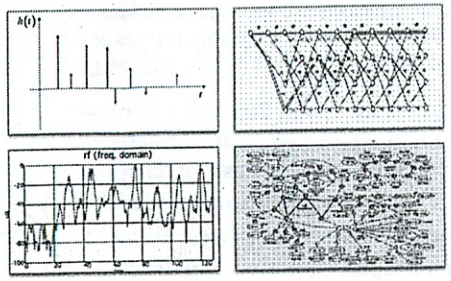
Can you verify this code has a Hamming distance of 3?

L well what are the code words?

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 $n \leq 2^{n-k} - 1$
  - Given k, use constraint to determine minimum n needed to ensure single error correction is possible 1 error
  - (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.



INTRODUCTION TO EECS II

DIGITAL COMMUNICATION SYSTEMS

6.02 Spring 2011
Lecture #9

Exam two double sided Crib sheet

- How many parity bits?
- Dealing with burst errors
- Reed-Solomon codes

4 data bits
+ 3 parity bits
7 bits

4	3

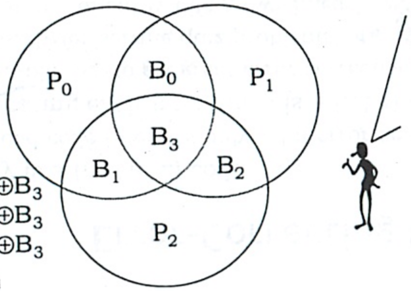
Single Error Correcting Codes (SECC)

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!

if more than 1 error this scheme fails

Checking the parity

Convert back
- recreate parity calculation

- 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:

$$E_0 = B_0 \oplus B_1 \oplus B_3 \oplus P_0$$
$$E_1 = B_0 \oplus B_2 \oplus B_3 \oplus P_1$$
$$E_2 = B_1 \oplus B_2 \oplus B_3 \oplus P_2$$

- If all the E_i are zero: no errors! Conclude message is correct
- Otherwise the particular combination of the E_i can be used to figure out which bit to correct.

if syndrome bit = 1 →

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?

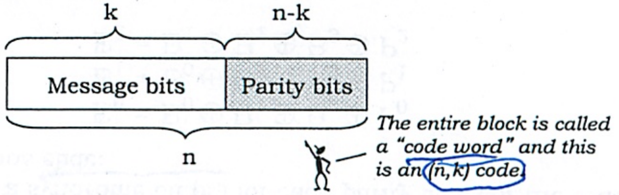
The 8 encodings indicate the 8 possible correction actions: no errors, error in one of 4 data bits, error in one of 3 parity bits

not just error detection but error correction

3/2

(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.

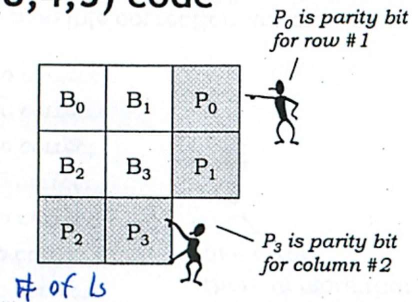
- Often we'll use the notation (n,k,d) where d is the minimum Hamming distance between code words. how much error correction it can do > 3
- The ratio k/n is called the *code rate* and is a measure of the code's overhead (always ≤ 1, larger is better).

Cable motens choose based on what it get backs

for vs its fixed

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

P_3 is parity bit for column #2

When not even the # of 1s

0 1 1
1 1 0
1 0
	0 1 1
	1 0 0 ← 2nd row
	1 0 ↖ intersection = flip
	↑ 2nd col
		0 1 1
		1 1 1 ← only in row so must be parity bit
		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?

$n+1 \le 2^{n-k}$

How many parity bits are needed?

- 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

  what need to encode
- 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 ≤ 2^{n-k} – 1 move 1 to other side
  - 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.

meet exactly the # of bits to use

Error-Correcting Codes

Each is different!

diff types of codes

- Parity is a (n+1,n,2) code
  - Good code rate, but only 1-bit error detection
- Replicating each bit r times is a (r,1,r) code
  - Simple way to get great error correction; poor code rate
  - Handy for solving quiz problems! to get code of Hamming of r
  - Number of parity bits grows linearly with size of message
- "Rectangular" codes with row/column parity
  - Easy to visualize how multiple parity bits can be used to triangulate location of 1-bit error
  - Number of parity bits grows as square root of message size
- Hamming single error correcting codes (SECC) are (n,n-p,3) where n = 2^p-1 for p > 1
  - See Wikipedia article for details
  - Number of parity bits grows as $\log_2$ of message size

pretty efficent

Noise models

- Gaussian noise
  - Equal chance of noise at each sample
  - Gaussian PDF: low probability of large amplitude
  - Good for modeling total effect of many small, random noise sources
- Impulse noise since not a single error
  - Infrequent bursts of high-amplitude noise, e.g., on a wireless channel
  - Some number of consecutive bits lost, bounded by some burst length B
  - Single-bit error correction seems like it's useless for dealing with impulse noise...

    or is it???

n blocks | n | n | n |

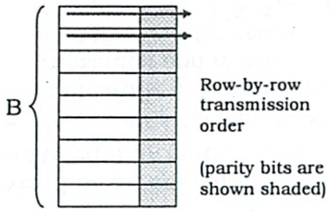
↓ wipes out block

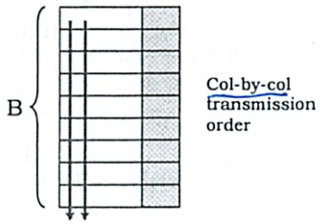
Dealing with Burst Errors

Correcting single-bit errors is nice, but in many situations errors come in bursts many bits long (e.g., damage to storage media, burst of interference on wireless channel, ...). How does single-bit error correction help with that?

how to deal w/ →

Well, can we think of a way to turn a B-bit error burst into B single-bit errors?





Problem: Bits from a particular code word are transmitted sequentially, so a B-bit burst produces multi-bit errors.

Solution: interleave bits from B different code words. Now a B-bit burst produces 1-bit errors in B different code words.

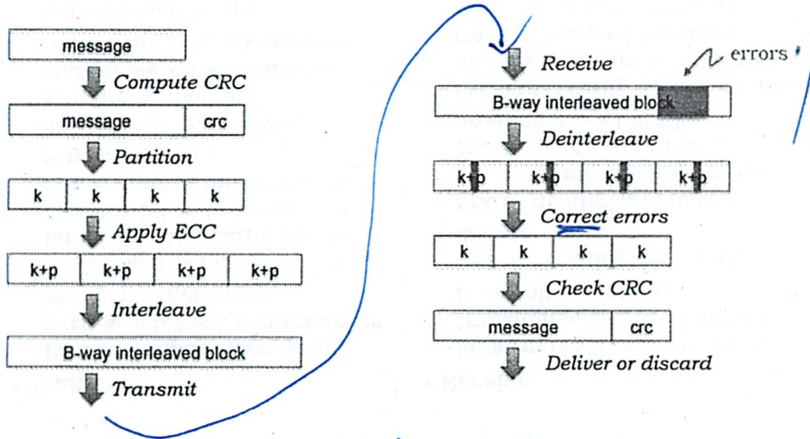
What if transmit all the first bits, 2nd bits, etc? "interleaving"

1111 2222 3333 4444

Can have burst up to B errors

Then 11 ~~22~~ 22 can be affected, so error correction works

Interleaving



But need to be careful where errors happen

Framing

Need to know where blocks start

- The receiver needs to know
  - the beginning of the B-way interleaved block in order to do deinterleaving
  - the beginning of each ECC block in order to do error correction.
  - Since the interleaved block is made up of B ECC blocks, knowing where the interleaved block begins automatically supplies the necessary start info for the ECC blocks

Our friend ↘

- 8b10b encoding provides what we need! Here's what gets transmitted
  - Prefix to help train clock recovery (alternating 0s/1s, ...)
  - 8b10b sync symbol
  - Packet data: B ECC blocks recoded as 8b10b symbols (after 8b10b decoding and error correction we get {#,data,chk})
  - Suffix to ensure transmitter doesn't cutoff prematurely, receiver has time to process last packet before starting search for beginning of next packet
  - On some channels: idle time (no transmission)

Reed-Sollomon codes

Our Recipe (so far)

- Transmit
  - Packetize: split message into fixed-size blocks, add sequence numbers, checksum
  - SECC: split {#,data,chk} into k-bit blocks, add parity bits to create n-bit code words with min Hamming distance of 3, B-way interleaving
  - 8b10b encoding: provide synchronization info to locate start of packet and sufficient transitions for clock recovery
  - Convert each bit into samples_per_bit voltage samples
- Receive
  - Perform clock recovery using transitions, derive bit stream from voltage samples
  - 8b10b decoding: locate sync, decode
  - SECC: deinterleave to spread out burst errors, perform error correction on n-bit blocks producing k-bit blocks
  - Packetize: verify checksum and discard faulty packets. Keep track of received sequence numbers, ask for retransmit of missing packets. Reassemble packets into original message.

bad throw away packet

Remaining agenda items

- With B ECC blocks per message, we can correct somewhere between 1 and B errors depending on where in the message they occur.
  - Can we make an ECC that corrects up to B errors without any constraints where errors occur?
  - Yes! Reed-Solomon codes
- Framing is necessary, but the sync itself can't be protected by an ECC scheme that requires framing.
  - This makes life hard for channels with higher BERs
  - Is there an error correction scheme that works on un-framed bit streams?
  - Yes! Convolutional codes: encoding and the clever decoding scheme will be discussed next week.

next week

In search of a better code

- Problem: information about a particular message unit (bit, byte, ..) is captured in just a few locations, i.e., the message unit and some number of parity units. So a small but unfortunate set of errors might wipe out all the locations where that info resides, causing us to lose the original message unit.
- Potential Solution: figure out a way to spread the info in each message unit throughout *all* the code words in a block. Require only some fraction good code words to recover the original message.

Spread information at through packet

~~Every~~ every bit depends on whole message

Thought experiment…

- Suppose you had two 8-bit values to communicate: A, B
- We'd like an encoding scheme where each transmitted value included information about both A and B
  - How about sending y = Ax + B for various values of x?
  - Standardize on a particular sequence for x, known to both the transmitter and receiver. That way, we don't have to actually send the x's – the receiver will know what they are. For example, x = 1, 2, 3, 4, … Come in a particular seq
  - How many values do you need to solve for A and B?
  - We'll send extra to provide for recovery from errors…

Oversampled polynomials

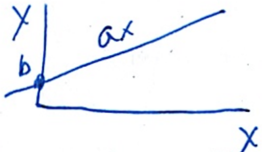
2 eq, 2 unknowns

coefficents are message bit

Send the results (the ys)

Can't just take first 2

Example



- Suppose you received four values from the transmitter y = 73, 249, 321, 393, corresponding to x = 1, 2, 3 and 4
  - 4 Eqns: $A \cdot 1+B=73$, $A \cdot 2+B=249$, $A \cdot 3+B=321$, $A \cdot 4+B=393$
- We need two of these equations to solve for A and B; there are six possible choices for which two to use
- Take each pair and solve for A and B

try each

$A \cdot 1+B=73$ $A \cdot 2+B=249$ *A=175, B=-102*	$A \cdot 1+B=73$ $A \cdot 3+B=321$ *A=124, B=-51*	$A \cdot 1+B=73$ $A \cdot 4+B=393$ *A=106.6, B=-33.6*
$A \cdot 2+B=249$ $A \cdot 3+B=321$ *A=72, B=105* ✓	$A \cdot 2+B=249$ $A \cdot 4+B=393$ *A=72, B=105* ✓	$A \cdot 3+B=321$ $A \cdot 4+B=393$ *A=72, B=105* ✓

← got 3 votes, the most, so use it

- Majority rules: A=72, B=105
  - The received value 73 had an error
  - If no errors: all six solutions for A and B would have matched

So long you don't get bad majority

Spreading the wealth…

- Generalize this idea: oversampled polynomials. Let

  $P(x) = m_0 + m_1x + m_2x^2 + \ldots + m_{k-1}x^{k-1}$

  where $m_0, m_1, \ldots, m_{k-1}$ are the *k* message units to be encoded. Transmit value of polynomial at *n* different predetermined points $v_0, v_1, \ldots, v_{n-1}$:

  $P(v_0), P(v_1), P(v_2), \ldots, P(v_{n-1})$

  Use any *k* of the received values to construct a linear system of *k* equations which can then be solved for *k* unknowns $m_0, m_1, \ldots, m_{k-1}$. Each transmitted value contains info about all m_i.
- Note that using integer arithmetic, the P(v) values are numerically greater than the m_i and so require more bits to represent than the m_i. In general the encoded message would require a lot more bits to send than the original message!

higher degree polynomial

Solving for the m_i

Solve for M 18.06

- Solving k *linearly independent* equations for the k unknowns (i.e., the m_i):

$$\begin{pmatrix} 1 & v_0 & v_0^2 & \cdots & v_0^{k-1} \\ 1 & v_1 & v_1^2 & \cdots & v_1^{k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & v_{k-1} & v_{k-1}^2 & \cdots & v_{k-1}^{k-1} \end{pmatrix} \begin{pmatrix} m_0 \\ m_1 \\ \vdots \\ m_{k-1} \end{pmatrix} = \begin{pmatrix} P(v_0) \\ P(v_1) \\ \vdots \\ P(v_{k-1}) \end{pmatrix}$$

- Solving a set of linear equations using Gaussian Elimination (multiplying rows, switching rows, adding multiples of rows to other rows) requires add, subtract, multiply and divide operations.
- These operations (in particular division) are only well defined over *fields*, e.g., rational numbers, real numbers, complex numbers -- not at all convenient to implement in hardware.

If use integer - can't do - lose info - the remainder
So must to irrational #

Finite Fields to the Rescue

- Reed's & Solomon's idea: do all the arithmetic using a finite field (also called a Galois field). If the m_i have B bits, then use a finite field with order 2^B so that there will be a field element corresponding to each possible value for m_i.
- For example with B = 2, here are the tables for the various arithmetic operations for a finite field with 4 elements. Note that every operation yields an element in the field, i.e., the result is the same size as the operands.

+	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

*	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

A	-A	A^{-1}
0	0	0
1	1	1
2	2	3
3	3	2

A + (-A) = 0

$A * (A^{-1}) = 1$

every arithmetic operation produces # that is member of field

How many values to send?

- Note that in a Galois field of order 2^B there are at most 2^B unique values v we can use to generate the P(v)
  - if we send more than 2^B values, some of the equations we might use when solving for the m_i will not be linearly independent and we won't have enough information to find a unique solution for the m_i.
  - Sending P(0) isn't very interesting (only involves m_0)
- Reed-Solomon codes use $n = 2^B-1$ (n is the number of P(v) values we generate and send).
  - For many applications B = 8, so n = 255
  - A popular R-S code is (255,223), i.e., a code block consisting of 223 8-bit data bytes + 32 check bytes

total message

Use for error correction

- If one of the $P(v_i)$ is received incorrectly, if it's used to solve for the m_i, we'll get the wrong result.
- So try all possible (n choose k) subsets of values and use each subset to solve for m_i. Choose solution set that gets the majority of votes.
  - No winner? Uncorrectable error… throw away block.
- (n,k) code can correct up to (n-k)/2 errors since we need enough good values to ensure that the correct solution set gets a majority of the votes.
  - R-S (255,223) code can correct up to 16 symbol errors; good for error bursts: 16 consecutive symbols = 128 bits!

They did not do the math actually

People at MIT fand a better way to actually solve

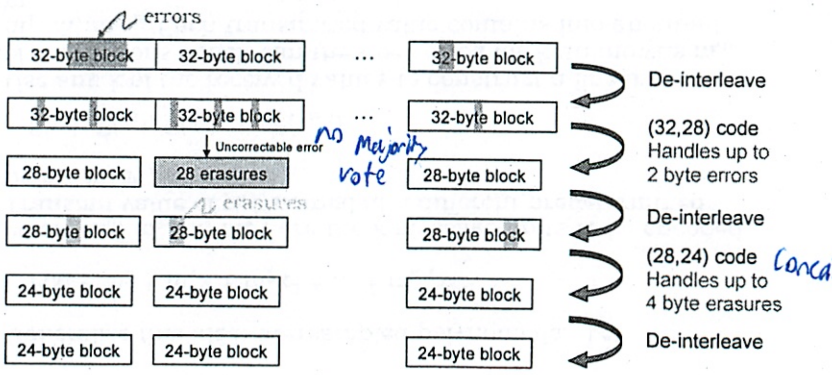
Erasures are special

just detect from matrix So can't corrupt voting

- If a particular received value is known to be erroneous (an "erasure"), don't use it all!
  - How to tell when received value is erroneous? Sometimes there's channel information, e.g., carrier disappears.
  - See next slide for clever idea based on concatenated R-S codes
- (n,k) R-S code can correct n-k erasures since we only need k equations to solve for the k unknowns.
- Any combination of E errors and S erasures can be corrected so long as 2E + S ≤ n-k.

Example: CD error correction

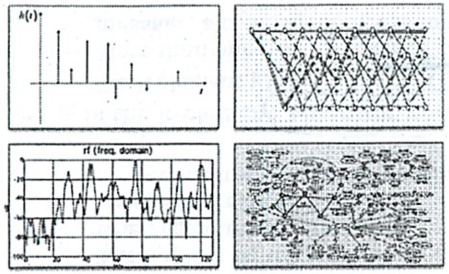
- On a CD: two concatenated R-S codes



no Majority vote

Concatinated

Result: correct up to 3500-bit error bursts (2.4mm on CD surface)



INTRODUCTION TO EECS II

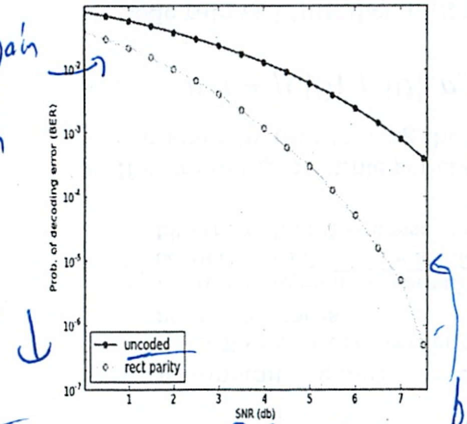
DIGITAL COMMUNICATION SYSTEMS

6.02 Spring 2011 Lecture #10

Quiz back Tue

- convolutional codes
- state & trellis diagrams
- most likely message to have been transmitted

Do We Need Better Channel Coding?



Some gain but not as much

↓ is better

↑ fn of x

non linear improvement

→ better

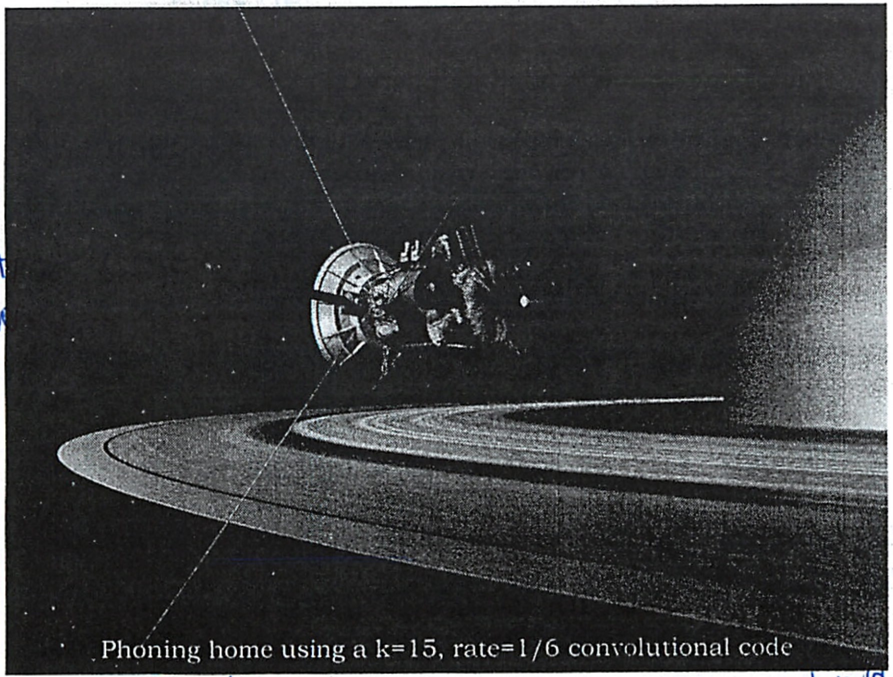
big improvement

The graph shows how a rate ½ "rectangular" block code experimentally improves over using no coding at all, especially at higher SNRs (lower overall BER).

But in low SNR environments, there's considerable room for improvement.

Can we find more effective rate ½ codes?

Complexity of reciever grows w/ k



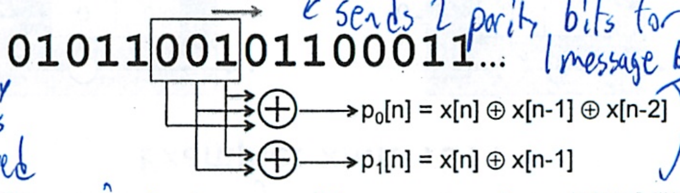
Phoning home using a k=15, rate=1/6 convolutional code

if retransmission is impractical — ~~takes~~ would months

So Use Convolutional code

Convolutional Codes

- Like the block codes discussed earlier, send parity bits computed from blocks of message bits
  - Unlike block codes, don't send message bits, only the parity bits!
  - The code rate of a convolutional code tells you how many parity bits are sent for each message bit. We'll be talking about rate 1/p codes.
  - Use a sliding window to select which message bits are participating in the parity calculations. The width of the window (in bits) is called the code's constraint length.



$p_0[n] = x[n] \oplus x[n-1] \oplus x[n-2]$

$p_1[n] = x[n] \oplus x[n-1]$

k = how many previous bits are involved

← sends 2 parity bits for every 1 message bit

rate = $\frac{1}{2}$

Might do multiple parity bits

Only send parity bits, not message bits

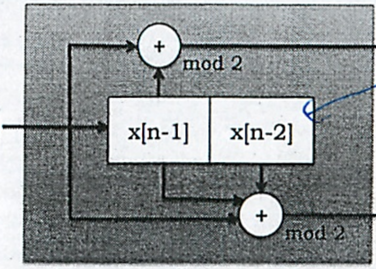
3/[illegible]

Block diagram view

hardware to do convolution codes

- One often sees convolutional encoders described with a block diagram like the following:

shift register-ish



The state

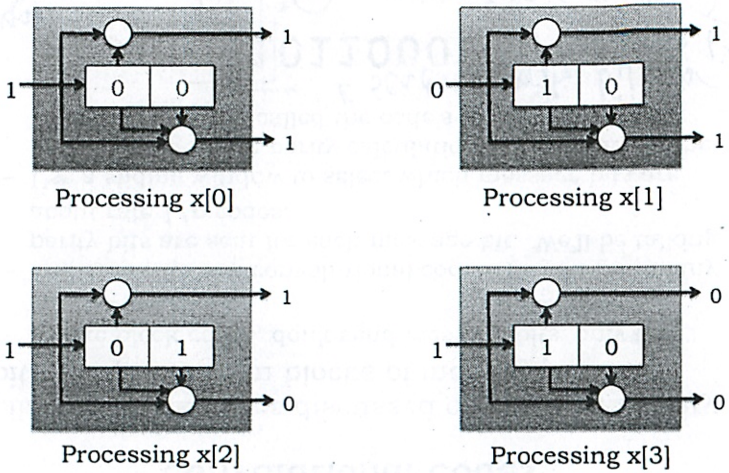
The x[n-i] values are referred to as the "state" of the encoder.

- Think of this a "black box": message in, parity out
  - Input bits arrive one-at-a-time on the wire on the left
  - The box computes the parity bits using the incoming bit and the k-1 previous message bits
  - At the end of the bit time, all the saved message bits are shifted right one location and the incoming bit moves into the left locn.

Some people ~~think~~ call this constraint length 3
← in this class " " 2

Same as moving window down string

Example: xmit 1011



Processing x[0]

Processing x[1]

Processing x[2]

Processing x[3]

Parity Bit Equations

- A convolutional code generates sequences of parity bits from sequences of message bits:

I can see why they call it a convolutional code

$$p_i[n] = \left(\sum_{j=0}^{k-1} g_i[j]x[n-j] \right) \bmod 2$$

- k is the constraint length of the code
  - The larger k is, the more times a particular message bit is used when calculating parity bits
    → greater redundancy
    → better error correction possibilities
- g_i is the k-element generator polynomial for parity bit p_i.
  - Each element $g_i[n]$ is either 0 or 1
  - More than one parity sequence can be generated from the same message; a common choice is to use 2 generator polynomials

Convolutional Codes (cont'd.)

- We'll transmit the parity sequences, not the message itself
  - As we'll see, we can recover the message sequences from the parity sequences
  - Each message bit is "spread across" k elements of each parity sequence, so the parity sequences are better protection against bit errors than the message sequence itself

Similar to Reed Solomon

- If we're using multiple generators, construct the transmit sequence by interleaving the bits of the parity sequences:

$$xmit = p_0[0], p_1[0], p_0[1], p_1[1], p_0[2], p_1[2], \ldots$$

interleave

- Code rate is 1/number_of_generators
  - 2 generator polynomials → rate = ½
  - Engineering tradeoff: using more generator polynomials improves bit-error correction but decreases the number of message bits/sec that can be transmitted

P-Set: punctured codes

k=3 (7,6) ← compact way of telling convolution code

↓

Example

↓ two 3 bit binary #

- Using two generator polynomials:
  - g_0 = 1, 1, 1, 0, 0, … abbreviated as 111 for k=3 code
  - g_1 = 1, 1, 0, 0, 0, … abbreviated as 110 for k=3 code
- Writing out the equations for the parity sequences:
  - $p_0[n] = (x[n] + x[n-1] + x[n-2]) \bmod 2$
  - $p_1[n] = (x[n] + x[n-1]) \bmod 2$
- Let x[n] = [1,0,1,1,…]; as usual x[n]=0 when n<0:
  - $p_0[0] = (1 + 0 + 0) \bmod 2 = 1$, $p_1[0] = (1 + 0) \bmod 2 = 1$
  - $p_0[1] = (0 + 1 + 0) \bmod 2 = 1$, $p_1[1] = (0 + 1) \bmod 2 = 1$
  - $p_0[2] = (1 + 0 + 1) \bmod 2 = 0$, $p_1[2] = (1 + 0) \bmod 2 = 1$
  - $p_0[3] = (1 + 1 + 0) \bmod 2 = 0$, $p_1[3] = (1 + 1) \bmod 2 = 0$

interleave

- Transmit: 1, 1, 1, 1, 0, 1, 0, 0, …

"Good" generator polynomials

Table 1-Generator Polynomials found by Busgang for good rate ½ codes

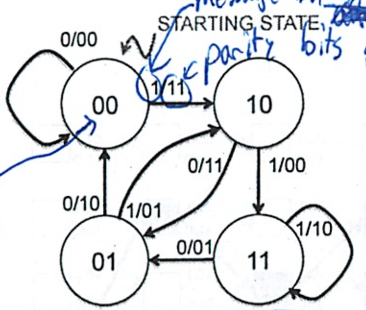
What shall I choose as a generator code?

Constraint Length	G_1	G_2
3	110	111
4	1101	1110
5	11010	11101
6	110101	111011
7	110101	110101
8	110111	1110011
9	110111	111001101
10	110111001	1110011001

www.complextoreal.com

One more way to represent the encoder

State Machine View



message bit to transmit

& parity bits going to output

register state entries

The state machine is the same for all k=3 codes. Only the p_i labels change depending on number and values for the generator polynomials.

- Example: k=3, rate ½ convolutional code
- States labeled with x[n-1] x[n-2]
- Arcs labeled with $x[n]/p_0p_1$
- msg=101100; xmit = 11 11 01 00 01 10

Message 1011

State	Msg	Parity	State'
00	1	11	10
10	0	11	01
01	1	01	10
10	1	00	11

2^{k-1} states

Pad out w/ 00 by convention

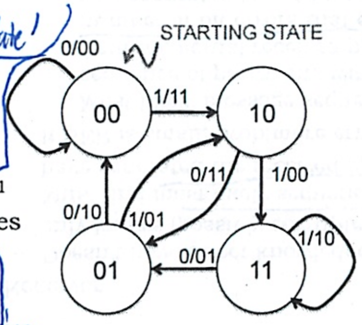
State	Msg	Parity	State'
11	0	01	01
01	0	10	00

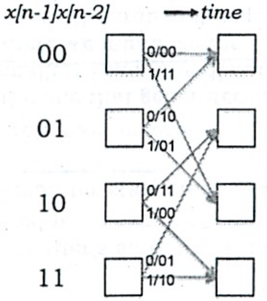
? to state 00 end

11 11 01 00 01 10 (down the column

State Machines & Trellises

another way to represent



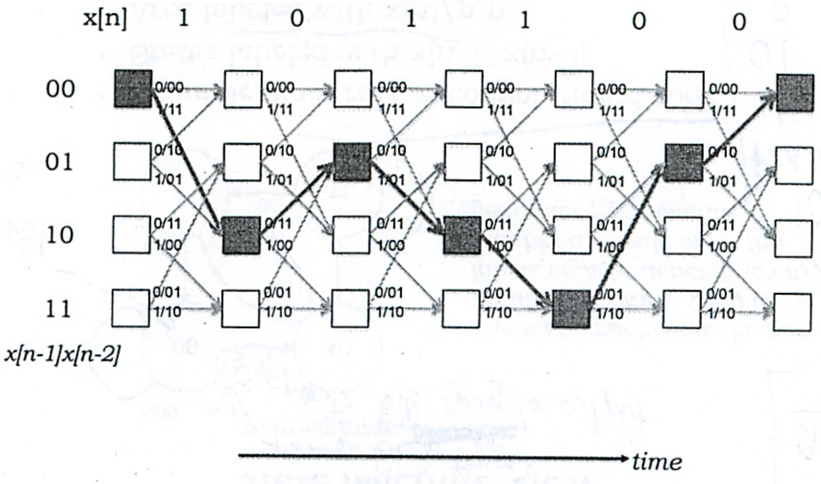


see next pg

- Example: k=3, rate ½ convolutional code
  - $G_0 = 111$: $p_0 = 1{*}x[n] \oplus 1{*}x[n-1] \oplus 1{*}x[n-2]$
  - $G_1 = 110$: $p_1 = 1{*}x[n] \oplus 1{*}x[n-1] \oplus 0{*}x[n-2]$
- States labeled with *x[n-1] x[n-2]*
- Arcs labeled with *$x[n]/p_0p_1$*

Addition mod 2 aka XOR

Trellis View @ Transmitter



Using Convolutional Codes

- Transmitter
  - Beginning at starting state, processes message bit-by-bit
  - For each message bit: makes a state transition, sends p_i
  - Pad message with k-1 zeros to ensure return to starting state
- Receiver
  - Doesn't have direct knowledge of transmitter's state transitions; only knows (possibly corrupted) received p_i
  - Must find most likely sequence of transmitter states that could have generated the received p_i
  - If BER is small, prob(more errors) < prob(fewer errors)
    - Most likely message sequence is the one that generated the sequence of parity bits with the smallest Hamming distance from the actual received p_i, i.e., where we minimize the number of bit errors that explains how the transmit sequence was corrupted to produce the received p_i

not very effective strategy

Example

assumed fewer errors more likely

- Using k=3, rate ½ code from earlier slides
- Received: 111011000110
- Some errors have occurred...
- What's the 4-bit message?
- Look for message whose xmit bits are closest to rcvd bits

Most likely: 1011

Msg	Xmit*	Rcvd	d
0000	000000000000	111011000110	7
0001	000000111110		8
0010	000011111000		8
0011	000011010110		4
0100	001111100000		6
0101	001111011110		5
0110	001101001000		7
0111	001100100110		6
1000	111110000000		4
1001	111110111110		5
1010	111101111000		7
1011	111101000110 10		2
1100	110001100000		5
1101	110001011110		4
1110	110010011000		6
1111	110010100110		3

*Msg padded with 2 zeroes before xmit

↓ Hamming Distance # positions where string bits don't match

but does not scale well to long messages

$P(\text{good}) = (1-BER)^6$

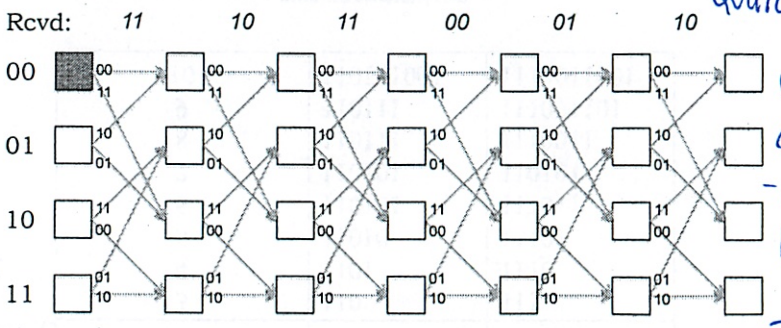
$P(1\text{ error exactly}) = \binom{6}{1}(1-BER)^5 \cdot BER$ ↑ fewer errors more likely

(assuming BER < .5) – which better be true

no guarentee

Finding the Most-likely Path

Viterbi – Qualcomm



used all over – maximum likely encoding

Given the received parity bits, the receiver must find the most-likely sequence of transmitter states, i.e., the path through the trellis that minimizes the Hamming distance between the received parity bits and the parity bits the transmitter would have sent had it followed that state sequence.

PRML algorithm to "guess" what is on HDD

6.02 Recitation

3/8

Error Correcting ~~Da~~ Code

- if bit errors, have a parity bit where can figure out what the bit error was + fix it

- 10 bit seq

  $2^{10} \simeq 1k$ - but one mistake and its screwed

  - what if want to send 1 message 0000000 ---
    1111111 ---

  - if get 000010000... - its more probable that you meant 00000000...

  - so you can flip the 1 to 0

* error correction comes from redundency

  L having ~~the~~ a structure - here repeating (which happens to be inefficent)

For us rate = 1/10

Its a tradeoff b/w bandwith (rate) and error correction

Rate	Allowable Errors for 10 bits
1/10	4 bits
1	0 bits

Recently a third metric: complexity

- power consumption

②

Can constrain possible sequences to send

- linear constraints - (in lecture]

  x_1, x_2, x_3 $\qquad x_i \in \{0, 1\}$

  $x_1 \oplus x_2 = 1$

  └ could be 100
  010
  011
  101

  lost something in rate
  but increased redunbency

- linear codes: (n, k, d_{min}) codes

  n = total # bits to send; message lenght

  k = # of possible code bits to transmit 2^k

  - first example: 2 possible codes $2^k = 2 \rightarrow k = 1$
    4 " " $2^k = 4 \rightarrow k = 2$

  d = hamming distance; quantities error correction ability
  - # of bits flipped b/w 2 codes

  r = rate = $\frac{k}{n}$

  d_{min} = min hamming distance ← this is what you specify

When you recieve a bit seq compare the hamming dist. Pick smallest

$$\begin{array}{l} 0000000000 \\ 0100001000 \\ 1111111111 \end{array} \begin{array}{l} \}\, 2 \leftarrow \text{fewer errors likley} \\ \}\, 8 \end{array}$$

③

$\left\lfloor \frac{d-1}{2} \right\rfloor$ ← floor

↑ max possible # errors where can still recover the message

Tutorial Qu

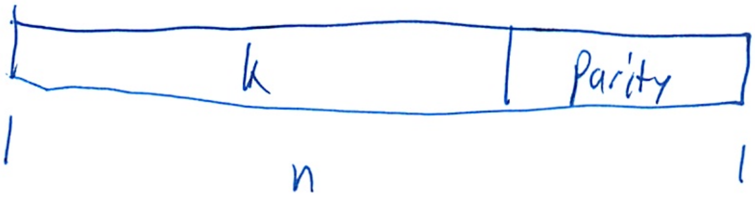
2. Error correcting code $(n, 20, 3)$

k ↑ need hamming distance of at least 3

Smallest n for this?

$d_{min} = 3$

$\left\lfloor \frac{d-1}{2} \right\rfloor = \left\lfloor \frac{3-1}{2} \right\rfloor = \left\lfloor \frac{2}{2} \right\rfloor = 1$ ← can deal w/ 1 error



parity bits should help remove errors

how many options to distinguish

1 = no error

n = 1 error in any pos

So $1+n \le 2^p$

↑ needs to be

If $k=2$ x_1, x_2

x_1	x_2	$x_1 \oplus x_2$

$p = x_1 \oplus x_2$

4

Use parity bits or syndrome

For every parity - calc parity you should have recieved

Then take diff parity recieved - parity calculated.

Should = 0

X_1	X_2	P_1
X_3	X_4	P_2
P_3		

$P_1 = X_1 \oplus X_2$

$P_2 = X_3 \oplus X_4$

$P_3 = X_1 \oplus X_3$

So want to transmit 1001 →

1	0	1
0	1	1
1		

X_1	X_2	X_3	X_4	P_1	P_2	P_3
1	0	0	1	1	1	1

Say error on recieving

1	0	0	1	0	1	1
Y_1	X_2	Y_3	Y_4	$\hat{P}_1$	$\hat{P}_2$	$\hat{P}_3$

Compute syndrome: ~~fon~~ try to compute parity from rec message bits

~~$X_1 \oplus X_2 = e_1$ expected $= 1$~~ ←

~~$X_3 \oplus X_4 = e_2$ $= 0$~~

~~$X_1 \oplus X_3 = e_3$ $= 0$~~

So

$Y_1 \oplus Y_2 = \tilde{P}_1$ ← calculated $\qquad \hat{P}$ = recieved

$Y_3 \oplus Y_4 = \tilde{P}_2$

$Y_1 \oplus X_3 = \tilde{P}_3$

Then Compare w/ what I. got

$$E_1 = \hat{P}_1 \oplus \tilde{P}_1 = 0 \oplus 1 = 1 \leftarrow \text{error}$$
$$E_2 = \hat{P}_2 \oplus \tilde{P}_2 = 1 \oplus 1 = 0$$
$$E_3 = \hat{P}_3 \oplus \tilde{P}_3 = 1 \oplus 1 = 0$$

↑ error

If more

$$2^p \geq \binom{n}{1} + \cdots + \binom{n}{t}$$

Shows how many errors you can correct

$$2^{n-k} \geq 1 + \binom{n}{1} + \cdots + \binom{n}{k}$$

6. Company says $n = 20$, $k = 16$ Can they correct at least 1 error(

$$2^{n-k} \geq n+1$$

$$2^4 \geq 21 \quad \otimes \text{ No!}$$

Look at tutorial problems for more clarification

3/8

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 PS4

Dates & Deadlines

issued:	Mar-02-2011 at 00:00
due:	Mar-10-2011 at 06:00
checkoff due:	Mar-15-2011 at 07:00

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.

For each of the following codes, indicate: (1) how many bit errors it is guaranteed to detect assuming only error detection is wanted and (2) how many bit errors it is guaranteed to correct assuming only error correction is wanted.

a. (10,1,10) code So this I has confused on

- replication code
will detect D-1 errors 9

(points: .33)

↑ is that just for that coding?

- yeah just detect that there is an/some sort of error

b. (153,132,7) code

Correction $\left\lfloor \frac{D-1}{2} \right\rfloor = 4$

Don't care what code detect $D-1=6$

correct $\lfloor\frac{D-1}{2}\rfloor=3$

(points: .33)

c. (15,11,3) Hamming code

2

1

(points: .34)

Problem 2.

what is hamming code vs hamming distance

↳ not caring details - type of SECC along w/ parity and replication

n,k,d

Suppose management has decided to use 48-bit message blocks in the company's new (n,48,3) error correcting code. What is the minimum value of n that will permit the code to be used for single bit error correction?

for any type of SECC

At minimum $n \leq 2^{n-k}-1$

$\hookrightarrow n = 2^{n-48}-1$

$n = 53.77 \rightarrow 54$

(points: 1)

where does d figure into this?

- tells us how many errors we can detect + fix $\lfloor\frac{3-1}{2}\rfloor = 1$ is lowest for 1 error

$D=3$

---

lots of diff stuff in these chaps - lots diff dimensions

Problem 3.

A set of five 4-bit data values has been encoded using the "(8,4,3)" rectangular parity code discussed in lecture and then transmitted over a noisy channel. For each of the received code words below indicate what single-bit error, if any, can be detected and corrected. The bits in the received code word are labeled as follows:

```
D1 D2 P1
D3 D4 P2
P3 P4
```

So calc syndrome

↓50

```
a. 0 0 1
   1 0 1
   0 0
```

Oh can do visually

(points: 0.2)

b.
```
1 0 1
0 0 0
0 0
```

Parity bit wrong

(points: 0.2)

this is fun!

c.
```
1 1 0
1 0 1
0 1
```

Correct

(points: 0.2)

d.
```
0 0 1
1 1 1
1 1
```

D=3 so can only fix 1 error

Does it work $n \leq 2^{n-k} - 1$

~~$4 \leq 2^{4-4} - 1$~~

Oh n = message + parity $8 \leq 2^{8-4} - 1$

$8 \leq 15$ ✓

Both wrong, retransmit

(points: 0.2)

e.
```
0 0 1
1 1 1
0 0
```

All parity wrong, retransmit

(points: 0.2)

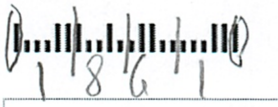
Problem 4.

As part of its efforts to automate mail delivery, the US Post Office often prints a bar code on each piece of mail as a way of encoding the destination zip code. The POSTNet code is based on a 2-of-5 code: there are five binary digits, exactly two of which are "1". To make it easy to read the code using an optical scanner, vertical lines of two different heights are used to represent 0 (short line) and 1 (tall line). Here's the code:

Zip code Digit	Encoding	Printed
1	00011	ıııll
2	00101	ıılıl
3	00110	ıullı
4	01001	ılııl
5	01010	ılılı
6	01100	ıllıı
7	10001	lıııl
8	10010	lııli
9	10100	lılıı
0	11000	llııı

two are always 1

a. When printing a POSTNet bar code, two tall vertical lines are added, one at either end. What digits does the following bar code depict (it's not a zip code):



(points: 0.5)

b. If we say that each zip code digit, encoded using 5 bits, is equivalent to 4 bits of information, what's the code rate of the POSTNet code? Please enter as a decimal fraction.

since some codes not included

Code rate = measure of overhead

$= \frac{k}{n}$ $\frac{4}{5}$

(points: 0.5)

c. For the POSTNet code (1) how many bit errors it can detect assuming only error detection is wanted and (2) how many bit errors it can correct assuming only error correction is wanted.

2 of 5
is m of n code
10 possible Combo
So good for # m of n

(5,4,—)

So depends on d
1 bit would show wrong
2? — needs to be automated way

— what is hamming distance of each with the other
— lot of combos — how many? — should find 2n-1?
— anyway 2

(points: 0.5)

d. Ben Bitdiddle, having just finished 6.02, has taken a VI-A internship at the Post Office. After studying the POSTNet code for a while, Ben writes an urgent memo to his supervisor suggesting the adoption of "binary zip codes" where zip code digits are constrained to be either "0" or "1". Ben acknowledges that the original 5-digit zip codes would become 17-bit binary zip codes, but he argues that using only the "0" and "1" encodings of the POSTNet code (the last and first entries of the above table, respectively) would enable much better error detection and correction.

For Ben's revised POSTNet code, write down: (1) how many bit errors it can detect, assuming only error detection is wanted; and (2) how many bit errors it can be guaranteed to correct, assuming error correction is wanted.

✓Hamming

So diff b/w 0 and 1 = 4

(points: 0.5) but stupid for other reasons – what are they?

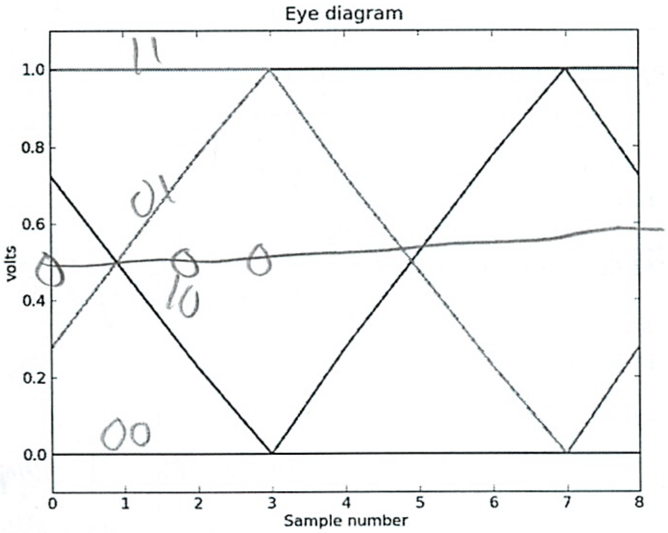
Python Task #1: Choosing the sampling point

more digits = more likly to be errors? no

Useful download links:

PS4_tests.py -- test jigs for this assignment
PS4_1.py -- template file for this task

Here's an eye diagram generated by transmitting a random sequence of bits across an idealized channel that limits the speed of transitions and the inter-symbol interference extends only only to the next bit. The transmitter uses 4 samples/bit and signaling voltages of 0.0V and 1.0V.



If we choose a digitization threshold of 0.5V, we can see that in this noiseless world we could successfully sample this signal using any sample numbered 0 mod 4, 2 mod 4, or 3 mod 4 -- i.e., only if we tried to sample the signal at samples numbered 1 mod 4 would we make mistakes in determining which bit the transmitter intended to send. But to make the

where it crosses

? how find algebraically?

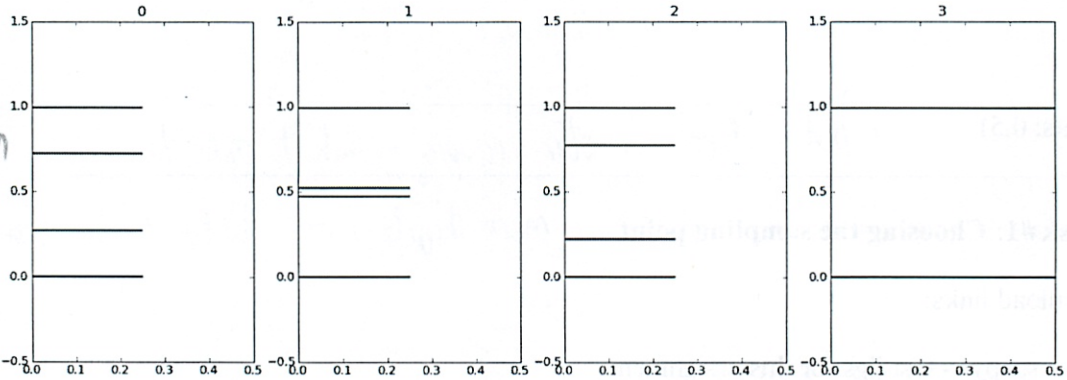
comparison with the digitization threshold as easy as possible, it would be best to choose the sample where the eye is most "open" -- samples numbered 3 mod 4 in this case.

In PSet #2, we used a clock and data recovery scheme that kept the sample point centered between the transitions. In this task we'll explore another approach to determining which receive sample to use.

is this related to what we learned?

As a first step in automating the process of determining where to sample, consider the following diagram which shows how the sample voltages are distributed at each of the four sample times within a bit cell. In the histogram for each of the four sample times, the length of the line indicates the fraction of samples that have the indicated voltage value.

Sideways histogram



At sample times 0, 1 and 2 the samples are divided evenly among four possible intermediate voltages; at sample 3 the samples are divided evenly between the two final voltages 0.0V and 1.0V.

Following the usual modus operandi, let's write a function to compute some statistics for each possible sample time given a particular channel. PS4_1.py is the template file for this task:

```
# this is the template for PSet #4, Python Task #1
import numpy
import PS4_tests

def sample_stats(samples,samples_per_bit=4,vth=0.5):
    # reshape array into samples_per_bit columns by as many
    # rows as we need.  Each column represents one of the
    # sample times in a bit cell.
    bins = numpy.reshape(numpy.array(samples),
                         (-1,samples_per_bit))

    # now compute statistics each column
    stats = []
    for i in xrange(samples_per_bit):
        column = bins[:,i]
        dist = column - vth  # subtract vth from each sample
        min_dist = ???  # your code here
        avg_dist = ???  # your code here
        std_dist = ???  # your code here
        stats.append((min_dist,avg_dist,std_dist))
```

4

1 | 2 | 3 | 4

? What are we doing?

So take 4 samples, See what min is

? how get that

Or do every fourth Sample

Oh they do that for us

all of the els of the ith value

```
        return stats  # return collected statistics

    if __name__ == '__main__':
        PS4_tests.test_sample_stats(sample_stats)

        stats = sample_stats(PS4_tests.channel_data)
        for i in xrange(len(stats)):
            min,avg,std = stats[i]
            print "sample %d: min_dist=%6.3f, avg_dist=%6.3f, " \
                  "std_dist=%6.3f" % (i,min,avg,std)
```

Finish the `sample_statistics` function given in the template so that for each of the possible sample times within a bit cell it prints the following:

`min_dist`
: For all the samples at this time, compute the voltage difference between each sample and the digitization threshold `vth`. Take the absolute value to measure the "distance" from the threshold and let `min_dist` be the minimum distance.

but where are we taking data from?
– samples

`avg_dist`
: For all the samples at this time, compute the voltage difference between each sample and the digitization threshold `vth`. Take the absolute value to measure the "distance" from the threshold and let `avg_dist` be the average of all the distances.

`std_dist`
: For all the samples at this time, compute the voltage difference between each sample and the digitization threshold `vth`. Let `std_dist` be the standard deviation of all the differences.

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

Oh it turned out to be very simple!

File to upload for Task 1: Browse...

That was a lot of figuring out code

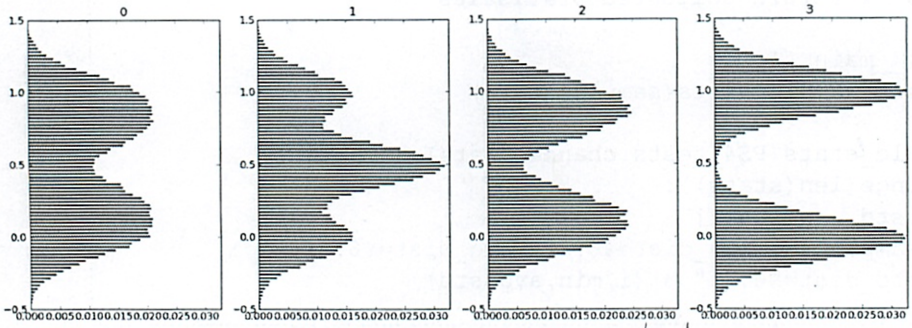
(points: 1)

What are the relative merits of using each of the statistics to determine which sample number corresponds to the most open part of the eye?

Pro Get largest diff   Con Change w/ time

(points: 1)

If we now add some noise to each channel (randomly chosen from a Gaussian distribution) and plot the sample distribution, the figure from above now looks like



noise!

With noise, at each sample time we no longer see a sample distribution that only has samples at a small number of voltages -- each of the lines in the first histogram has been replaced by a Gaussian distribution centered where the lines used to be.

Rerun your code, this time calling `sample_stats` with the argument `PS4_tests.noisy_channel_data`. Think about the results and select the statistic you'll use to determine which sample number corresponds to the most open part of the eye.

Cut and paste the output of your Task #1 code running on the noisy channel data into the answer box below. Then explain why the `min_dist` statistic is no longer a good choice for determining optimal sample number when the channel adds noise to the channel.

all 0

Use avg instead

(points: 1)

Python Task #2: Determining the bit error rate.

Useful download link:

PS4_2.py -- template file for this task

Using the results of your deliberations in Task #1, write a Python function `receive` that returns the recovered sequence of message bits given a vector of voltage samples produced by the channel:

`message_bits = receive(samples,samples_per_bit=nsamples)`

First apply the statistical measure you chose in Task #1 to the *`samples`* array to determine which sample time in the bit cell should be used for determining the transmitted message bit. Digitize that sample in each cell and return the resulting sequence of message bits.

Now write a second Python function `bit_error_rate` that compares two bit sequences and returns the fraction of bit locations that don't match. For example, if two 1000-element bit sequences mismatch in two bit locations, the result would be 2/1000 = .002.

float

error_rate = bit_error_rate(*seq1*,*seq2*)

Return the fraction of bit locations that don't match between the two locations.

Finally write a function `compute_ber` that returns an estimate of the bit error rate given a digitization threshold of `vth`, assuming Gaussian noise with a mean of 0 and a variance of σ^2.

ber = compute_ber(*sigma*,vth=0.5,v0=0.0,v1=1.0,p0=0.5,p1=0.5)

Return estimate for bit error rate given the *sigma* of the Gaussian noise. Use the supplied voltages for the digitization threshold and the means of the voltages for 0 bits and 1 bits.

`vth` is the digitization threshold, defaults to 0.5V. `v0` is the mean of voltages received for 0 bits, defaults to 0V. `v1` is the mean of voltages received for 1 bits, defaults to 1V.

`p0` and `p1` are the probabilities of transmitting 0 bits and 1 bits respectively. You can assume they sum to 1.

just compute math

You'll find it useful to call `PS4_tests.unit_normal_cdf(x)` which returns the area under the curve for the unit normal, integrating between -∞ and x.

PS4_2.py is a template for testing your functions using a million-bit message:

```
# this is the template for PSet #4, Python Task #2
import matplotlib.pyplot as p
import math,numpy,random
import PS4_tests

def receive(samples,samples_per_bit=4,vth=0.5):
    """
    Apply a statistical measure to samples to determine which
    sample in the bit cell should be used to determine the
    transmitted message bit.  vth is the digitization threshold.
    Return a sequence or array of received message bits.
    """
    pass # your code here

def bit_error_rate(seq1,seq2):
    """
    Perform a bit-by-bit comparison of two message sequences,
    returning the fraction of mismatches.
    """
    pass # your code here

def compute_ber(sigma,vth=0.5,v0=0.0,v1=1.0,p0=0.5,p1=0.5):
    """
    Return an estimate of the bit error rate given the values
    for the threshold voltage and the two received voltages
    for 0 and 1.  Use PS4_tests.unit_normal_cdf if you need
    values of Φ(x).
    """
    pass # your code here


if __name__ == '__main__':
```

```
# make sure functions pass some simple tests
PS4_tests.test_bit_error_rate(bit_error_rate)
PS4_tests.test_compute_ber(compute_ber)

# construct a test message
message = [random.randint(0,1) for i in xrange(1000000)]

# try out different noise levels
ber_values = []
snr_values = []
for sigma in (0.5,0.25,0.18,.05):
    noisy_data = PS4_tests.transmit(message,
                                    samples_per_bit=4,
                                    nsigma=sigma)
    received_message= receive(noisy_data)
    ber = bit_error_rate(message,received_message)
    ber_values.append(ber)

    # use 0.25 as power of signal (see lec. slides)
    snr = 10*math.log(0.25/(sigma**2),10)
    snr_values.append(snr)

    print "For sigma = %g" % sigma,
    print "(SNR = %g db):" % snr,
    print "bit error rate = %g," % ber,
    print "computed BER = %g" % compute_ber(sigma)

"""
# plot BER vs SNR
p.figure()
ax = subplot(111)
p.plot(snr,ber,'b-',lw=2)
p.title('BER vs SNR')
ax.set_yscale('log')
ax.set_ylabel('BER')
ax.set_xlabel('SNR (db)')
p.show()
"""
```

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

File to upload for Task 2: Browse...

(points: 4)

Please cut and paste the output of your Task #2 code in the answer box below. Explain why would you would expect a small difference between the experimental BER and the computed BER.

Randoness

(points: 1)

Python Task #3: Error correction

Useful download link:

PS4_3.py -- template file for this task

In this task, your job is to take a received codeword which consists of a data block organized into `nrows` rows and `ncols` columns, along with even parity bits for each row and column. The codeword is represented as a binary sequence (i.e., a list of 0's and 1's) in the following order:

```
[D(0,0), D(0,1), ..., D(0,ncols-1),          # data bits, row 0
 D(1,0), D(1,1), ...,                        # data bits, row 1
 ...
 D(nrows-1,0),    ..., D(nrows-1,ncols-1),   # data bits, last row
 R(0), ..., R(nrows-1),                      # row parity bits
 C(0), ..., C(ncols-1)]                      # column parity bits
```

? how to do matrix math?

in other words, all the data bits in row 0 (column 0 first), followed all the data bits in row 1, ..., followed by the row parity bits, followed by the column parity bits. The parity bits are chosen so that all the bits in any row or column (data and parity bits) will have an even number of 1's.

Define a Python function `correct_errors(g)` as follows:

message_sequence = `correct_errors(`*codeword,nrows,ncols*`)`

codeword is a binary sequence of length `nrows*ncols + nrows + ncols` whose elements are in the order described above.

first chop off parity bits

The returned value *message_sequence* should have `nrows*ncols` binary elements consisting of the corrected data bits D(0,0), ..., D(nrows-1,ncols-1). If no correction is necessary, or if an uncorrectable error is detected, just return the raw data bits as they appeared in the codeword.

`PS4_tests.even_parity(`*seq*`)` is a function that takes a binary sequence *seq* and returns True if the sequence contains an even number of 1's, otherwise it returns False. This parity check will be useful when performing the parity computations necessary to do error correction. PS4_3.py is a template for testing your function:

how to flag where error? a map?

Or go through each bit and check its parity
- not with this code really

```
# template for PSet #4, Task #3
import PS4_tests

# return data portion of codeword (nrows*ncols bits) with errors
# corrected.  If uncorrectable error, return raw bits.  codeword
# is a binary sequence that starts with the data row-by-row,
# followed by row parity bits, followed by column parity bits.
def correct_errors(codeword,nrows,ncols):
    # the raw data bits
    data = codeword[0:nrows*ndata]
```

do one on paper   Or just bad rows array

```
        # do row & column parity checks, correct indicated error
        # ... YOUR CODE HERE...

        # return the posibly corrected data
        return data

    if __name__ == '__main__':
        PS4_tests.test_correct_errors(correct_errors)
```

The `PS4_tests.test_correct_errors` function will try a variety of test codewords and check for the correct results. If it finds an error, it'll tell you which codeword failed; if your code is working, it'll print out "Tests completed successfully."

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

File to upload for Task 3: Browse... ✓

(points: 5)

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

That was bit of pain
- last tests took 1 hr

1

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

Col = 4
row = 2

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

Parity bits seperated correct

row - not correct

Is it when odd - since works....

Oh row/col with that bit added will be even!

Oh that Numpy no append thing!

Can it only fix one error?

- yeah

- What if error in only 1 row
  - Then parity bit error

Then how to test position

ncol * colerror + row error

2

$$\begin{array}{cc|cl} 1 & 0 & 0 & \leftarrow \\ 1 & 1 & 0 & \\ 0 & 0 & 0 & \\ 0 & 0 & 0 & \\ \hline 0 & 0 & & \\ & \uparrow & & \end{array}$$

n cols · bad col + bad row

2 - 1

Oh loads new when one passes

$$\begin{array}{cc|cll} 1 & 1 & 0 & & \\ 0 & 1 & 0 & \leftarrow & 1 \\ 0 & 0 & 0 & & \\ 0 & 0 & 0 & & \\ \hline 0 & 0 & & & \\ \uparrow & & & & \\ 0 & & & & \end{array}$$

Shall 3

(ncols-1) · bad ~~col~~ row

~~(2-1) · 1~~

2 · 1

passed!

n cols · badrows + bad cols