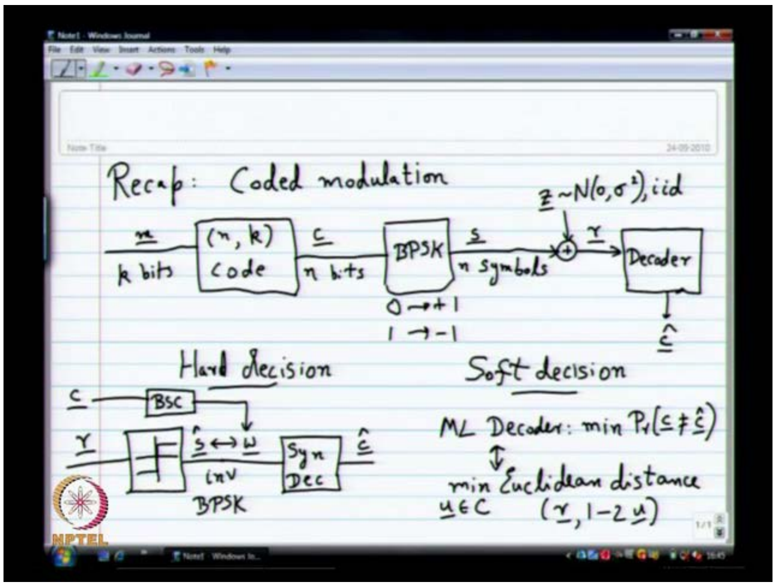
Coding Theory
Dr. Andrew Thangaraj
Department of Electronics & Communication Engineering
Indian Institute of Technology, Madras

Lecture - 19
Bitwise Map Decoder for BPSK over AWGN

(Refer Slide Time: 00:14)



So, let us begin with a quick recap is also a minor bug in notation which I will fix in this recap itself. So, the basic problem we are dealing with here is coded modulation, so the last time, I was talking in at length about why this is different from simple error correction. Why people moved to this coded modulation idea and how it is implementable today in systems and what are the possible benefits.

So, we did not really quantify the benefits may be we will quantify that in this lecture, so what is the picture, now the picture basically looks like this you have a k bit message m which gets encoded by an let say using an n comma k code. Something is going on here n comma k code, so suddenly it seem to be having difficulty writing, so let me rewrite again, sorry about that you have an n k code. It is converted this into a code word of n bits and then the difference is we will also consider the modulation block which converts bits into what I am going to call as symbols.

Essentially, these symbols represent did not think we use s, so these symbols represent signals that are going into the channel, so you use a basis for those signals and decompose it and we will use only a one dimensional situation. So, you have BPSK which takes 0 to plus 1, suddenly this touch screen is behaving in a peculiar fashion that is 0 goes to plus 1 and 1 goes to minus 1. So, this is my simple BPSK conversion, to this I am adding Gaussian noise which is normal with mean 0 and variance sigma square.

We get a receive vector r and essentially encoded modulation, we will run a decoder that works on r and produces let say a c cap and estimate of estimate of c. So, this is the idea, so your decoder works with real valued vector as suppose to what we had before which was essentially a binary vector or say a vector over g f q or g f p. So, this is a big difference and basically we distinguish between two cases one, so I think one was the hard decision situation.

Then, the others were soft decision, so I do not know if my handwriting has become bad or it is screen is trying to be too smart. If there like a piece of cloth, I can rub I think it is wet because of that I am some difficulty and usually it is not so bad. I can write and its little bit legible, so I am going use my handkerchief and clean this screen assumable that is the problem, so usually it is not. So, you guys agree it is not this bad, it is not the best looking handwriting out there, but it is not terrible from a legitimate or I mean being legible point of view.

So, we distinguish between two cases hard decision and soft decision, so on hard decision the first block that you do after you receiver is what I called a slicer. So, you slice it and this is a symbol by symbol slicer and you produce some kind of an estimate of s cap which is basically equivalent to somewhat I called as w. This w is basically from c you can have a binary symmetric channel and you can get w, so that was the idea. So, you s cap would be a symbol it will be plus one or minus 1 from plus 1 you go to 0 and minus you go to 1, so this is basically inverse BPSK you do inverse BPSK, you are you are simulating a binary symmetric channel. Then, you run your classical syndrome decoder on this you run a syndrome decoder you it is it is m l, it is an m l decoder for the binary symmetric channel and you get a c hat.

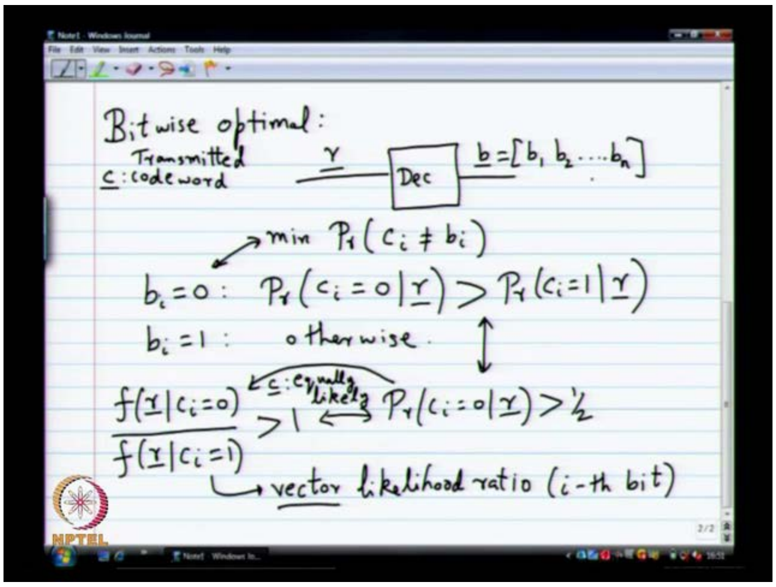
So, this is hard decision, but clearly you can see already that is because even though the vector r is not i i d it is not i i d this is treating as i i d at least for the slicing point of view from a slicing point of view you should not be slicing it, but it is slicing it. So, this is the hard decision decode in soft decision we at this point, we think in last class we defined m l decoder which minimizes the block error probability that c is not equal to c cap and I showed the equivalence in BPSK over AWGN and m l decoder is equivalent to minimizing Euclidean distance.

So, I am briefly sketching this, so hopefully you remember what it is Euclidean distance between r and r and 1 minus 2 u, so that is what the m l decoder is and this is true in general and for the BPSK k s. You can also show this is the same as maximizing the dot product between r and 1 minus 2 u, so those are various versions of the m l decoder, but I quickly point it out. I give you a slightly non trivial example and I kind of pointed out that it is going to become very difficult as n becomes large. So, you going to get n k becomes large you have to do a lot of correlations lot of computations completely in practical and to implement.

So, you cannot do that, so of course, what is actually implemented in practice often times is an approximate soft decision decoder something for which you can prove any optimality results, but you can analyze it maybe in some in some manner. You can analyze it in some approximate manner; you can analyze it and show using simulations that it works very well. So, that is how people do today, so it seems like pretty ad hoc solutions to an engineering problem, but that is pretty much how these approximate soft decoders are studied.

So, that is something to keep in mind, but before that we have to definitely see the optimal decoders, so know what the optimal stuff is before you go ahead and approximate it. Otherwise you will never continuing to approximate without knowing whether there is anything optimal out there or not, so the next type of optimality which I said is slightly easier to get to in practice is the bitwise m a p decoders.

(Refer Slide Time: 08:57)



Bitwise optimal:

Transmitted $\underline{c}$: codeword

$$\xrightarrow{\underline{r}} \boxed{\text{Dec}} \longrightarrow \underline{b} = [b_1\, b_2 \dots b_n]$$

$$\min P_r(c_i \neq b_i)$$

$b_i = 0 : \; P_r(c_i = 0 | \underline{r}) > P_r(c_i = 1 | \underline{r})$

$b_i = 1 : \;$ otherwise.

$$\frac{f(\underline{r}|c_i=0)}{f(\underline{r}|c_i=1)} > 1 \xleftrightarrow{\underline{c}:\text{ equally likely}} P_r(c_i = 0 | \underline{r}) > \tfrac{1}{2}$$

$\hookrightarrow$ $\underline{\text{vector}}$ likelihood ratio (i-th bit)

So, bitwise optimal decoders, so in this case you think of your decoder working on r as producing some vector b and the goal is to minimize probability that c i not equal to b i what is c i c i is the code word transmitted code word. So, I was this is kind of where we stopped or we did this towards the end of last class and I made a mistake in the notation. So, I am going to correct that now, so I was able to claim that this would be the same as what I made a claim which is which we do not prove.

I think it is quite easy if you go the whole analysis this is the same as maximizing what i mean, I think I have maybe approached little bit differently. I think I approached it little bit differently, so let me just do that once again, sorry about that. So, I said basically to do this you have to say b i is 0 if probability that b i oh c i, sorry that is why I made a mistake.

I put b i inside, I should have put c i there, c i equal 0 given the entire vector r is greater than probability that c i equal to 1 given the entire vector r. So, this is the rule which I said is equivalent to this so that you can prove using meaning some very basic analysis. I am not going to do it, it is kind of intuitive and clear that is what you should do and compute the probability let us greater you decide b i is 0 and you decide b i equal to 1.

So, quite a simple rule and then we define the modified rule, we looked at several equivalent ways of doing it and one thing I claim was this is the same as probability that c i equal to 0 given r is greater than half. It is the same as that that is a very easy thing to see and this is also the same as the what I called as the likelihood ratio which is f of r given c i is 0 divided by is there a question. So, f of r given c i is one is greater than 1, so in this step you have to assume equally likely messages, so you have to say c is equally likely.

So, that is important and also if also make the assumption that the code is code does not have some trivial construction I mean the code you should not have one bit which is always 0. So, that is a very weird construction to have, normally you should not have that as long as you do not have this. So, this is this is good enough and this I called as the vector likelihood ratio for the i th bit or so vector because we have looking at the entire vector r, this is likelihood ratio, this is for the i th bit.

So, I think this is pretty much where we have stopped in the last lecture, so is a good point to continue once again,, so are there any questions on these, this is reasonably clear.

Student: Sir, because we are doing by bit mean c i by I th bit to another for different bits differently, so we can say that the bit outcome may not be code word.

Yes something like that, so the question was is the outcome guarantee to be a code word I made a comment yesterday that the outcome need not be a code word. So, this b need not necessarily be a code word, but if you are doing systematic encoding clearly the first k bits of this out vector b, you can take it as your decoded message. So, that is it is not a problem you can always so that, but the output is definitely not guarantee to be a code word that is the rule that I am using and the rule I am only trying to minimize probability that c i is not equal to b i.

So, I am not trying to force the constraint that the vector b has to be from some set doing it just bit by bit without any without any other constraint.

Student: Sir, how is this different from hard decision.

How is that different from?

Student: Using our idea using in vector in them.

How is it how is what different from hard decision?

Student: Ah the result from this can be different from.

Yes, the r here is the continuous real valued vector that you received, so the question was why this is different from hard decision. So, maybe as I do it and we will see an example and you will see clearly is not hard decision. Before that, even let me confused about why you are asking me the question why do you think it should be same as the hard decision what would not what depend just on f of the entire vector r given c i. How many of you think the entire vector r given c I will depend only on the i th bit what do you mean by depending on the i th bit.

So, the question is suppose i is 1, I am looking at c 1 r 1 is effected only c 1 or let say is r 2 not effected by c 1. Let me ask that question is that question prove or not or will think about it let say you go from r 1 to r k and then you go to r k plus 1. One can argue that r 1 through r k maybe is affected only by the c 1s, if you do systematic encoding you picking c 1 through c k independently. Definitely r k plus 1 will depend on all c 1 through c k r k plus 2 will depend on c 1 through c k again it will depend also on will c k plus 1 given c k plus 1 obviously it would not depend on anything else, but I am not giving c k plus 1.

Here, I am only giving c i, so we will write down precise expressions here means this is a this is an important question because most people will see it for the first time get confused by this. This is the standard thing and probability difference between conditional independence and independence so two random variables or two events can be conditionally independent given some other event. In totality, they may not be independent at all, so this is one of those cases clearly it happens sampled value of r at that.

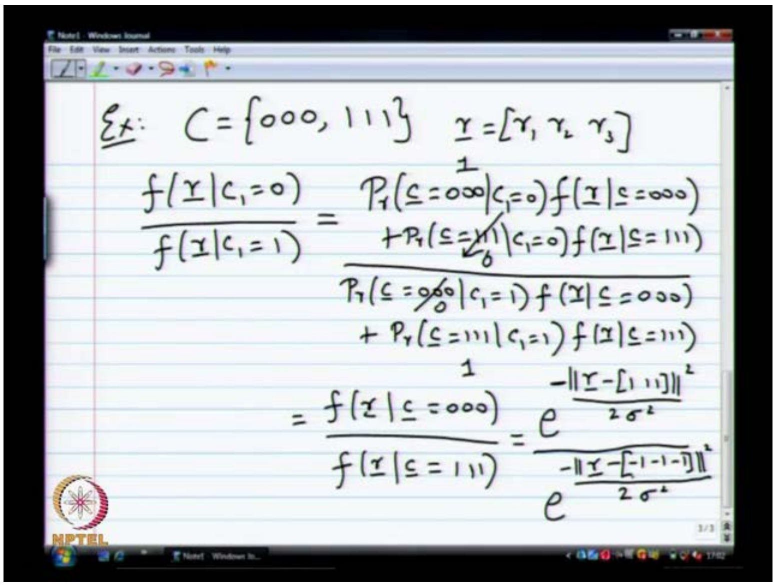
So, r i is not looking at r, this is entire vector r, that is why I have once again emphasized this is vector likelihood ratio, I am sorry yes how do we compute it, let us see when you know all the probability tools to compute this. I will do it explicitly in some examples and general also we will do it, so we will do it will see how to compute right but but the question that you are asking is important. So, r a does not it depend only on c i, you should ask when that is that is the question that you should ask it is true.

So, it depends on c i, so given c i r i does not depend on any other code word bit that is true given c i, but let me complete this is very important, let say r i does not depend on any other c j given c i. So, that is that is something that is something that is important, but now we have done if you look at the entire vector r given c i, you cannot say the entire vector is independent of everything else. Of course, you have so many other things also, so that is what that is what you have to look at very careful and be sure about.

So, given c i alone does not make all the r s independent does not happen if you are given the entire code word, then all the r s are independent, but only c i is given. Then, all the vectors all the values in r does not do not become independent think about it for a while it is an important thing to be convince about because the entire decoder hinges on that.

Anyway, when I write the expression, it will be more clear to you will write an explicit expression for this and will see clearly some values terms will occur which are not independent. You will see how it works out, so let us write that down, so let us take a very simple example first and write this expression. Then, we will go on and see more complicated examples and then I will write the more general most general form of it, it is very easy, it is very basic elementary probability.

(Refer Slide Time: 18:36)



$\underline{Ex}$: $C = \{000, 111\}$ $\quad \underline{r} = [r_1\ r_2\ r_3]$

$$\frac{f(\underline{r}|c_1=0)}{f(\underline{r}|c_1=1)} = \frac{\overset{1}{P_r(\underline{c}=000|c_1=0)} f(\underline{r}|\underline{c}=000) + P_r(\underline{c}=\underset{0}{\cancel{111}}|c_1=0) f(\underline{r}|\underline{c}=111)}{P_r(\underline{c}=\underset{0}{\cancel{000}}|c_1=1) f(\underline{r}|\underline{c}=000) + \underset{1}{P_r(\underline{c}=111|c_1=1)} f(\underline{r}|\underline{c}=111)}$$

$$= \frac{f(\underline{r}|\underline{c}=000)}{f(\underline{r}|\underline{c}=111)} = \frac{e^{\frac{-\|\underline{r}-[1\ 1\ 1]\|^2}{2\sigma^2}}}{e^{\frac{-\|\underline{r}-[-1\ -1\ -1]\|^2}{2\sigma^2}}}$$

It is bit confusing, so that is why you should see this very carefully, so in the example we will take the first the 3 comma 1 repetition code. So, let us try to write down f of r given c 1 equals 0 divided by f of r given c 1 equals 1, so here you will get a slightly different answer which is not general, but anyway let me just write it down. Then, I will see something bit more different, so remember r is a general vector r 1 r 2 r 3 how do you evaluate this probability is important how do you how will I write the numerator what can I use, how can I express the numerator in any terms.

Of course, you have make use of that knowledge, so the only knowledge you have you do not have any other knowledge c 1 is zero implies what c 2 is also 0 c 3 is also 0, so it is a very simple code. So, c 1 is 0 means the entire code word was 0, 0, 0, so the way you write this is basically you have to say this is probability that c equals 0, 0, 0 given c 1 equal 0 which is actually 1. Anyway, I am writing it down just to show you how it works times f of r given c equals 0, 0, 0, so that is how you write the numerator see from c 1 you go to c because c 1 is 0, c is constraint to be.

So, let me let me write it down, I am sorry let me write down the whole thing, then will we will see how it works sorry so basically the best way of writing it. So, you have to say probability that c equals 0, 0, 0 given c 1 equal 0 times f of r given c equals 0, 0, 0 plus probability that c equals 1, 1, 1 given c 1 is 0 f of r given c equals 1, 1, 1. Then, the denominator you write a very similar expression, so what is happening is 1 is given c 1 is 0 as c 1 is 1 from c 1 you have to go to c, there is a probability, then from c you have to go to r from c 1 you are going to the vector c.

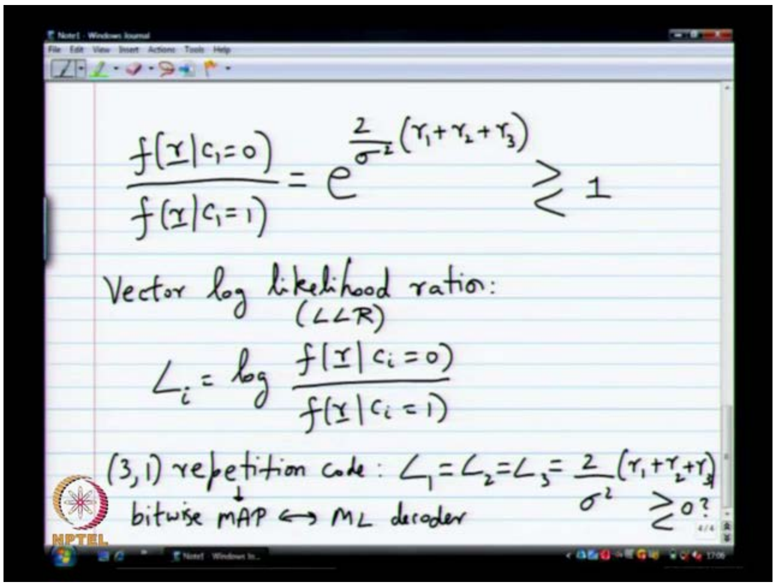
Then, from c you are going to the vector r, so the probability distribution from going from the vector c to the vector r is well known to you. It is all independent products of several Gaussians what is the probability distribution from going from going from c 1 to the vector c that is where the knowledge of the code comes in c 1 is 0. What do I know about these probabilities, this guy is 0 and this guy is 1, likewise here this guy is 0 and this guy is 1 that is why I use my knowledge of the code in more complicated situation. There will be more terms coming in here it is just only one term it is a very trivial code, so you get the answer very easily. So, this basically becomes f of r given c equals 0, 0, 0 divided by f of r given c equals 1, 1, 1.

Now, some of you will argue I can directly like this without any in this case it seems like a very simple step. I agree with you it is simple in this case, but I write this down I wrote this down in such a laborious fashion because the same formula can be repeated for any code. We will see the probability there will change it is very easy to do that, so what is this expression you know now that r has become conditionally independent you have given all the code word bits. So, the vectors of r have now become conditionally independent.

So, you simply multiply the exponentials that you get from the Gaussian distribution the constants will go to are going to cancel in the numerator and the denominator. So, you only have to worry about what is in the exponent and the exponent also you know how it is going to look. It is simply going to be the distance between r and 1, 1, 1 and r and minus 1, minus 1 minus 1, so that is what you will get, so essentially this is going to be e power minus r minus 1, 1, 1 square divided by 2 sigma square. The whole thing divided by e power minus 1, minus 1, minus 1 is that is it 2 sigma square is that. So, I have an explicit formula for by vector likelihood ratio in terms of the receive vector r.

So, once you give me an r, I will go plug the plug it and compute this and see which is whether it is greater than 1 or not. In fact, we can simplify this little bit more after all this is just a square right sum of square if you simplify it a little bit more, let us see what you get in this case.

(Refer Slide Time: 25:12)



$$\frac{f(\underline{r}|c_1=0)}{f(\underline{r}|c_1=1)} = e^{\frac{2}{\sigma^2}(r_1+r_2+r_3)} \gtrless 1$$

Vector log likelihood ratio:
(LLR)

$$L_i = \log \frac{f(\underline{r}|c_i=0)}{f(\underline{r}|c_i=1)}$$

(3,1) repetition code : $L_1 = L_2 = L_3 = \frac{2}{\sigma^2}(r_1+r_2+r_3) \gtrless 0?$

$\downarrow$

bitwise MAP $\longleftrightarrow$ ML decoder

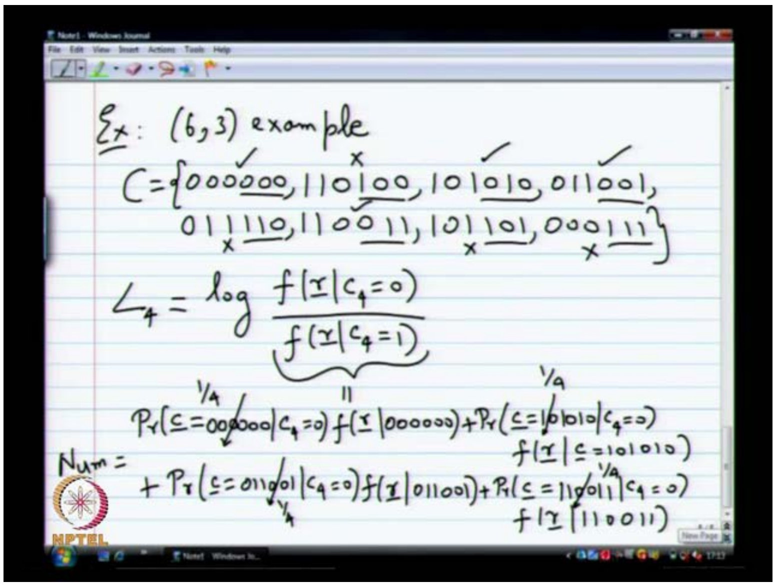
You can show f of r given c i is 0 divided by f of r given c 1 is 1 will work out to what we do the simplification it show it will become e power 2 by sigma square times r 1 plus r 2 plus r 3. You can do that work it is just basic algebra, you see you can cancel it out and subtract it and you will get this answer 2 by sigma square into r 1 plus r 2 plus r 3. So, one thing you are noticing when you do this likelihood ratios is everything is e power something. So, it makes a lot of sense to take logarithms for the likelihood ratio, anyway you are going to only check if this is greater than 1 or less than 1, so it maybe take logarithm since c if it is greater than 0 or less than 0.

So, it will be totally equivalent, so that is why people usually use the log likelihood ratio as suppose to the as suppose to just likelihood ratio for doing anything. So, we will first define this vector log likelihood ratio which is basically in a in a general setting. It is this log likelihood ratio is abbreviated as l l r, so the vector l l r will be log of f of r given c i is 0 divided by f of r given c i is 1. So, we can call it maybe since it is a vector we call it capital l i, so this is the vector l l r for the i th bit using the i th bit. So, in fact for the repetition code if i change this c 1 to c 2 what will happen, it would not make any difference because all the 3 bits are the same you will get the exact same answer.

So, for the repetition code all the three vector likelihoods are exactly the same and they are all equal to this. So, for the 3, 1 repetition code L i equals L 2 equals L 3 equals 3 by sigma squared times r 1 plus r 2 plus r 3 and how do I decide on the particular bit I simply see if this quantity is greater than 0 or less than 0. What do we have here actually the same as m l, so for the repetition code the bitwise optimal decoder matches with the m l decoder. So, it will definitely only output the code word, so you do not have to worry about it, so it is a good code on that sense what we will see other codes are not guaranteed to do such things.

So, we will have all kinds of other answers, so the repetition codes the 3, 1 repetition code bitwise MAP decoder is the same as the m l decoder, so the condition here is either this is greater than 0 or less than 0. So, I want to take a slightly more complicated situation and let us go back to our standard example, I want to do the 6, 3 example I know it is little bit more complicated.

(Refer Slide Time: 29:12)



Ex: (6, 3) example

$C = \{\overset{\checkmark}{000\underline{000}}, \overset{\times}{110\underline{100}}, \overset{\checkmark}{101\underline{010}}, \overset{\checkmark}{011\underline{001}}, \underset{\times}{011\underline{110}}, 1100\underline{11}, \underset{\times}{101\underline{101}}, \underset{\times}{000\underline{111}}\}$

$$L_4 = \log \frac{f(\underline{r} \mid c_4 = 0)}{\underbrace{f(\underline{r} \mid c_4 = 1)}}$$

$$\text{Num} = \overset{1/4}{Pr}(\underline{c} = 000000 \mid c_4 = 0) f(\underline{r} \mid 000000) + \overset{1/4}{Pr}(\underline{c} = 101010 \mid c_4 = 0) f(\underline{r} \mid \underline{c} = 101010) + \underset{1/4}{Pr}(\underline{c} = 011001 \mid c_4 = 0) f(\underline{r} \mid 011001) + \overset{1/4}{Pr}(\underline{c} = 110011 \mid c_4 = 0) f(\underline{r} \mid 110011)$$

I think it is just within reach of working by hand and it will really illustrate some of the ideas that are crucial. So, let us see the 6, 3 example let me see we need we really need the list of code words, so let me try and do this of the top of my head. I think it is 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0. Then, 0 1 1 0 0 1 then four 1 0, then 0 0 1 1 1 0 1 1 0 1 and finally, 0 0 0 1 1 1, so this is the 6, 3 code. So, this is the same code that we have been using all along from day one, I think maybe we are using this code.

So, let us try and write down some expressions here, so remember again are going to think of the last three bits as the systematic bits. So, it makes sense to try and compute L 4. So, when I do bitwise MAP decoder, it is enough if I find the message bits right why will i bother finding the parity bits. It is a good question to ask, but maybe you should worry about parity bits also, if you are going to implement the bitwise MAP decoder brute force by computing the expression fully.

You do not really have to find the parity bits; just find the message bits, so if you want to find the parity bits it is simply do an encoding again with the message bits. You found you can find the parity bits if you want means that is another way of doing things which is guaranteed to give you a code word. By the way, that is one more way of thinking about let us do L 4 and if L 4 remember is f of r given. So, once again I mean the setting I am not repeating the setting it is always the same you doing BPSK and AWGN and you get a receive vector r which will which in this case will have 6 different coordinates r 1, r 2 r 3 r 4 r 5 r 6.

So, I am asking this question, so there is a logarithm in front of this I have really erase it and write it again sorry about log, this log will usually take as log base e. So, remember that it is logarithm base e that is why the e power just goes away, so f of r given c 4 is 0 divided by f of r given c 4 is 1. Now, when I do the do the computation, you will not simply get a 0, 1 like you had before you will have multiple expressions in the numerator and multiple expressions in the denominator. So, you have to pay attention to that, so let us let us let us write this ratio then we will the logarithm finally, so it is not a big deal the likelihood ratio becomes. So, there will be probability that c equals what are the possibilities for c given that c 4 is 0.

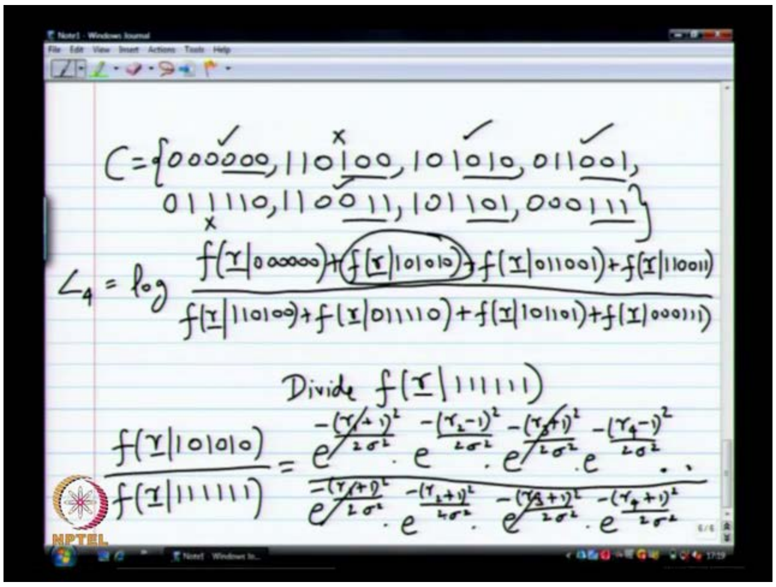
So, you have this guy, this one this one and this one, so it is enough if you only look at these four possibilities and you can also assume all those are equally likely because I said the message all the code words are equally likely. So, you can assume these four are equally likely the others will have probability 0, so each of these four will have probability one fourths. So, we will come to that slowly enough, so people are making some comments about what will happen in the general situation I am glad you are thinking about that, but let us first do this.

Then, we will go to that be a little bit more patient, so you have to write down I will write the first numerator alone in in full expansion in a boring way. Then, we will simplify and quickly write the final answer, so the first expression given c 4 is 0 times f of r given I will write 0 0 0 0 0 0 plus probability that c is 1 0 1 0 1 0 given times f of r given c equals 1 0 1 0 1 0 plus probability. That c equals 0 1 1 0 0 1 given four is 0 times f of r given 0 1 1 0 0 1 plus probability that c equals, so this is the numerator, remember c equals 1 1 0 0 1 1 given c 4 is 0 times f of r given 1 1 0 0 1 1.

So, this is how the numerator will look and each of these terms will have probability 1 by 4. So, likewise when you write the denominator you will have how many possibilities four possibilities what are the possibilities that show up in the denominator 1 2 3 4 exactly the other four. Exactly, the other four will show up and once again you will have the one fourth, one fourth, one fourth, one fourth, multiplying the conditional p d fs. So, you can cancel all those one fourths away when you take the ratio and write an expression only involving the conditional p d fs in the numerator.

There will be four terms and then in the denominator there will be the four other terms. So, that is what I am going to do in the in the next page, but then I will have to remember the code, so hopefully I remember, let us see if I do not remember I will come back and fix it. So, let us try some of the more fancy thing this is suppose to work let me see we try some of the more fancy there, we go copy there we go it is not bad at all, so that is the code.

(Refer Slide Time: 36:50)



$$C = \{\overset{\checkmark}{000\underline{000}}, \overset{\times}{110\underline{100}}, \overset{\checkmark}{101\underline{010}}, \overset{\checkmark}{011\underline{001}}, \underset{\times}{011110}, 110\underline{011}, 101\underline{101}, 000\underline{111}\}$$

$$L_4 = \log \frac{f(\underline{r}|000000) + \boxed{f(\underline{r}|101010)} + f(\underline{r}|011001) + f(\underline{r}|110011)}{f(\underline{r}|110100) + f(\underline{r}|011110) + f(\underline{r}|101101) + f(\underline{r}|000111)}$$

Divide $f(\underline{r}|111111)$

$$\frac{f(\underline{r}|101010)}{f(\underline{r}|111111)} = \frac{\cancel{e^{-\frac{(r_1+1)^2}{2\sigma^2}}} \cdot e^{-\frac{(r_2-1)^2}{2\sigma^2}} \cdot \cancel{e^{-\frac{(r_3+1)^2}{2\sigma^2}}} \cdot e^{-\frac{(r_4-1)^2}{2\sigma^2}} \cdots}{\cancel{e^{-\frac{(r_1+1)^2}{2\sigma^2}}} \cdot e^{-\frac{(r_2+1)^2}{2\sigma^2}} \cdot \cancel{e^{-\frac{(r_3+1)^2}{2\sigma^2}}} \cdot e^{-\frac{(r_4+1)^2}{2\sigma^2}}}$$

So, L 4 then becomes, so let me write this down f of r given 0 0 0 0 0 0 f of r given 1 0 1 0 1 0 plus f of r given 0 1 1 0 0 1 plus f of r given 1 1 0 0 1 1 is that. So, that is the numerator, then the denominator you will have f of r, so log I forgot is as usual thanks for reminding me. So, L 4 is there is a log then you have r given 1 1 0 1 0 0 plus f of r given 0 1 1 1 1 0 plus f of r given 1 0 1 1 0 1 0 0 0 1 1 1. So, all this is a bit boring and I want to also emphasize the fact that doing optimal bitwise soft decoders and all quite boring exercises. So, it is not so easy you have to look at all possibilities even, so this is just for the 6, 3 code imagine what will happen if you do a 1000 comma 500.

So, it is just impossible, these exercises are very difficult and the approximate methods that will introduce later we will introduce later. They will do these things in an iterative manner approximately and that turns out to be good enough so keep that in mind. So, do not lose hope as we as we write these painful expressions, but there is some light at the end of the tunnel. So, remember each of these expressions how will each of these expressions look like it will be a product of several Gaussians and because it is a r becomes conditionally independent once you give the code word, so what we can do there is a nice bit of tricky.

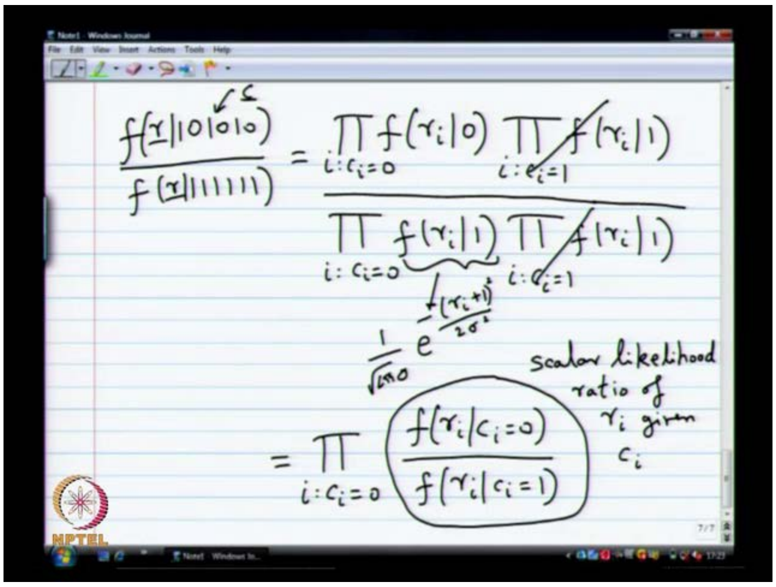
You can do here it is not very great it is still the expression is still messy, but at least you will write it in some other terms which will give you a lot of intuitions about how this is working. So, maybe it will give you some intuitions, so what you do is you divide numerator and denominator by this term, I know all of you are going to object that this is not a code word. I know this is not a code word, but this is some expression and I can definitely divide numerator and denominator by this expression. Nothing stop me from doing that so divide each term by this expression, so you will get is whenever there is a one what will happen to that term.

It will cancel it out whenever there is a 0, it will divide by that corresponding received value given one, so what you will get effectively when you divide by this is you will get this scalar log likelihood ratios in the numerator and denominator always. So, you will get an expression when you are going from this scalar likely log likelihood ratios or scalar likelihood ratios to the vector likelihood ratios. So, I am going to show how that happens here, so maybe for one term and then you will see for how it works in general. So, let us pick this term this term here and divide by this and see what happens, so f of r given 1 0 1 0 1 0 divided by f of r given 1 1 1 1 1 1.

So, this is going to be the constant, I will ignore, so it is anyway gets we have got rid of it. This will be e power minus r 1 plus 1 squared by 2 sigma square times e power minus r 2 minus 1 squared by 2 sigma square times e power minus r 3 plus 1 squared by 2 sigma square. I am going to run out of room, whenever I run out of room, I will stop, so hopefully you can extrapolate what is there, so 4 minus 1 squared by 2 sigma squared. Then, you will have two other terms, I am not writing the two other terms, I shall be two other terms you can imagine what the two other terms will be. Then, dividing by what e power minus r 1 plus 1 whole squared by 2 sigma squared times e power r 2 plus 1 whole squared by 2 sigma squared e power minus.

So, there are various other ways of writing it, but anyways let me just go through this painfully first time. Then, you see clearly there is some cancellation whenever you have A 1 it will cancel whenever you have A 0 it will not cancel, so essentially another way of writing this which will maybe simplify things a lot is f.

(Refer Slide Time: 42:18)



$$\frac{f(\underline{r}|\overset{c}{101010})}{f(\underline{r}|111111)} = \frac{\prod_{i:c_i=0} f(r_i|0) \prod_{i:c_i=1} \cancel{f(r_i|1)}}{\prod_{i:c_i=0} \underbrace{f(r_i|1)}_{\frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(r_i+1)^2}{2\sigma^2}}} \prod_{i:c_i=1} \cancel{f(r_i|1)}}$$

$$= \prod_{i:c_i=0} \boxed{\frac{f(r_i|c_i=0)}{f(r_i|c_i=1)}}$$

scalar likelihood ratio of r_i given c_i

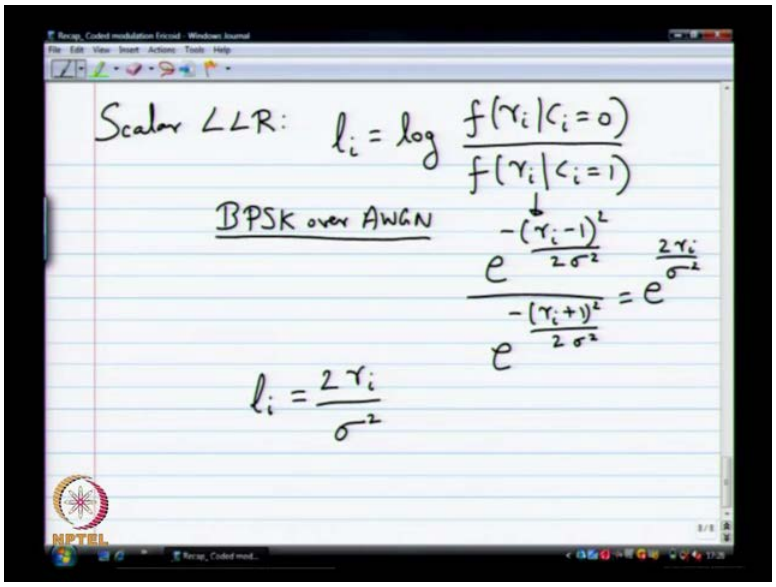
Let me write down the same expression again and in a slightly different way f or r given 1 0 1 what is the expression, I am sorry 1 0 1 0 1 0 divided by f of r given 1 1 1 1 1 1. This is basically the product f of r i given what is such that c i is 0 i such that c i is 0 suppose this is this vector is some c f of r i given 0. Then, you also have a product i such that c i is 1 f of r i given 1 and the denominator what you have all of them are 1s. So, product over all i, so let me write it down also just like that even though the denominator is different I will write it r I given.

So, this is essentially what is happening here, so remember what is this what is this notation it is clear what this notation is it is basically that expression e power one by root pi sigma. So, this what you will get one by root two pi sigma e power minus r i plus 1 squared by 2 sigma square. So, that is the expression, so clearly what will happen is this guys will cancel wherever you have 1s that term will cancel and wherever you have 0 will get this ratio. Now, new ratio which is again a likelihood ratio likelihood ratio of only the scalar r i given c i so that is what you get.

So, essentially what you get here is product of i such that c i is zero this ratio f of r i given c i is 0 divided by f of r i given c i is 1. So, this quantity we will call as small l i which is this, so this quantity we will call as e power small l i which is basically the scalar likelihood ratio.

Then, I will denote as l i the logarithm of this, so that is what I am going to do of r i given c i, so that is important because I am only talking about r i. Now, given c i r i just depends on c i nothing else, so I know I am about this laboring this point a little bit, but I think it is important that you see that the vector log. Likelihood ratio depends only on the scalar log likelihood ratio is nothing else, so the ratios is what will matter, so let us introduce this notation we will use this scalar l l r.

(Refer Slide Time: 45:45)



Scalar LLR: $l_i = \log \dfrac{f(r_i|c_i=0)}{f(r_i|c_i=1)}$

BPSK over AWGN

$$\frac{e^{\frac{-(r_i-1)^2}{2\sigma^2}}}{e^{\frac{-(r_i+1)^2}{2\sigma^2}}} = e^{\frac{2r_i}{\sigma^2}}$$

$$l_i = \frac{2r_i}{\sigma^2}$$

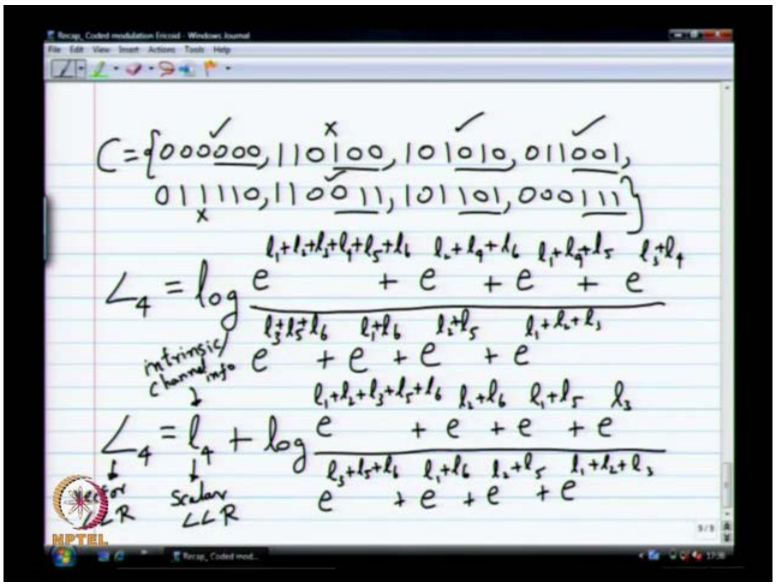
This is basically small l i it is the logarithm of f of r i given c i zero divided by f of r i given c i is 1. So, for BPSK over AWGN it will cancel very nicely what will happen for BPSK over AWGN this product will simply be e power minus r i minus 1 whole square by 2 sigma square divided by e power r i plus 1 whole square by two sigma square. So, this basically become e power 2 r by sigma squared, so you can simplify 2 r i by sigma square is that so for BPSK over AWGN for BPSK over AWGN.

I am not able to write suddenly, let me try once again for BPSK over AWGN it is not working the mouse seems to be working, but did somebody do something here actually it is not even coming down here. I do not know why just stops there can you see that it is stopping there it is not coming down below this. I have to restart mouse button probably, so it is works out like this, so in when you take logarithm what will happen you will get 2 r i by sigma square. So, for BPSK over AWGN the scalar l l rs have this very simple expression which is 2 r i by sigma squared the i th received value is simply scaled by 2 by sigma square.

You get the scalar log likelihood ratio, so remember this is only for the i th position given the i th bit r i given c i that is all what you really want is the vector l l rs that is what you really want. If you have that, then you can make a decision on the i th bit, now there is a question about why is this bitwise MAP is different from hard decision decoding in hard decision decoding. You use the scalar l l s scalar l l rs to make decisions on make decisions on each bit, but that is not your final decision. It is only a tentative decision because after that you are again going to do syndrome decoding, but in bitwise MAP.

You do not even do that, you do not even make a decision first and then do syndrome decoding. You directly compute the vector l l r and then use that to find the decision on the bit clearly finding the vector l l r. You saw how messy the expression is not the same as doing syndrome decoding after scalar l l r quantization. So, it is not it will not be the same you can see that it is not same, so let us go through the previous example use this idea of scalar l l rs. Then, try and write down the entire expression, so let me see if it has the code in memory it still has it, so if you want to write l 4.

(Refer Slide Time: 49:41)



$$C=\{\overset{\checkmark}{000\underline{000}},\ \overset{\times}{110\underline{100}},\ \overset{\checkmark}{101\underline{010}},\ \overset{\checkmark}{011\underline{001}},\ \underset{\times}{011}110,\ 110\underline{011},\ 101\underline{101},\ 000\underline{111}\}$$

$$L_4=\log\frac{e^{l_1+l_2+l_3+l_4+l_5+l_6}+e^{l_2+l_4+l_6}+e^{l_1+l_4+l_5}+e^{l_3+l_4}}{e^{l_3+l_5+l_6}+e^{l_1+l_6}+e^{l_2+l_5}+e^{l_1+l_2+l_3}}$$

intrinsic/channel info

$$\underset{\substack{\downarrow\\ \text{vector}\\ \text{LLR}}}{L_4}=\underset{\substack{\downarrow\\ \text{Scalar}\\ \text{LLR}}}{l_4}+\log\frac{e^{l_1+l_2+l_3+l_5+l_6}+e^{l_2+l_6}+e^{l_1+l_5}+e^{l_3}}{e^{l_3+l_5+l_6}+e^{l_1+l_6}+e^{l_2+l_5}+e^{l_1+l_2+l_3}}$$

I know I have to worry about code words with fourth position 0 and the numerator and then code words with fourth positions 1 in the denominator, but then I am going to divide by something to make everything into l l rs. So, that is what I am going to do, so all I have to do is something like this, I will logarithms whenever I have 0, I should have a e power small l i term. Whenever I have A 1, I do not have to have that that is that is how each of the terms will look like, so I am going to write down the final expression. It will be like this e power the first one will be what all the terms will come.

So, that will be l 1 plus l 2 plus l 3 plus l 4 plus l 5 plus l 6 what about the next term e power l 2 plus l 4 plus l 6 what about the next term e power l 1 plus l 4 plus l 5 and what about the last term e power l 3 l 4. Then, you divide is that what do you get in the denominator e power l 3 l 5 l 6 plus e power l 1 l 6 plus e power l 2 l 4 plus e power l 1 l 2. I think I should put plus, I am sorry forgetting the plus everywhere l 1 plus l 2 plus l 3 is that.

So, one thing you notice is e power l 4 occurs in every term in the numerator in fact that will happen always if I compute some L i power small l i will show up in every term in the numerator. I am including only those code words which have c i equal 0 in the numerator, so clearly that term will show up in every single term. So, I can pull out e power l 4 and then I have log of a b which is log a plus log b log of e power l 4 is simply small l 4. So, you will get if you do the simplification here small l 4 plus log of something, what will be that something you can write that down.

So, I am going to write it down once again just to be plus e power l 2 plus l 6 plus e power l 1 plus l 5 e power l 3 plus l 4 divided by e power. So, denominator does not change, thank you very much plus e power l 1 plus l six plus e power l 2 plus l 4 plus e power l 1 plus l 2 plus l 3. So, this expression to me brings out a lot of nice properties about the bitwise m a p decoder and in fact this expression is also general. If I instead of 4 if I put 5 what will happen, here I will definitely will get a small l 5 plus log some numerator some denominator fourth terms in the numerator fourth terms.

In the denominator, several e power l sums, so that is how it will look in general this is a general thing. So, we also generalize it soon enough and I will write a very general looking expression will see that this is true. So, this is some kind of a scalar l l r, this is the vector l l r, so when you compute the vector l l r accurately the scalar l l r is the starting point. So, it is some kind of an intrinsic information this thing is also called intrinsic information, so scalar l l r i can this is intrinsic or channel information.

So, this is in fact l 4 has nothing to do with the code itself, you are looking at r four and computing the likelihood for c four without taking into account the coding at all so it is just intrinsic to that channel whatever the channel is telling you about c 4. At that point, you are only using the channel information; likewise you have all the small l s which are all intrinsic channels information about each of those bits you have to combine them in this peculiar way not peculiar straight for way.

You can say one can say straight forward, but it is a complicated way to get the capital l four which is really what you want which is the your decision statistics based on that you make a decision and this combination is obviously very complicated. There is clearly no easy way to see that this is the same as hard decision decoding, so obviously it would not be for say it would not be you will believe me.

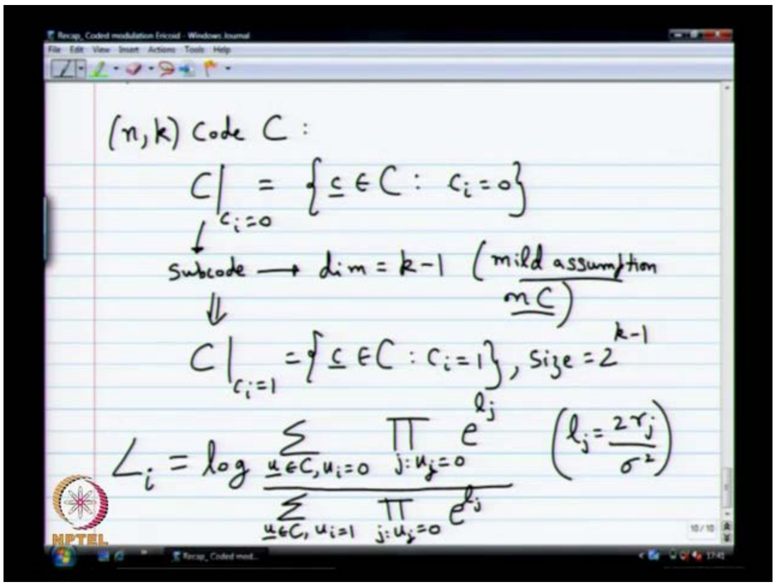
If I say it is the same as hard decision I have really prove something, so it is impossible to prove well actually we are not neglecting, so in we are doing it differently in hard decision what are we doing instead of using l four in all its entirety. We are saying we will only instead of using the small l s in all their real number complexities we are only using one bit about them when you do a hard decision using l 4, what are you doing using r 4 when you do a hard decision you are saying. I am only keeping one bit of information about the entire r four here we are using all the bits that is the difference between hard decision and soft decision.

After you take one bit you are still doing some processing like this you are doing a syndrome decoding right and then only you are deciding. So, there is something some processing happening, but it is not the same as using the entire real valued information about r 4 is this throwing everything away. You are keeping only one bit of information, so for instance if you quantize all the small l s to just one bit and run your bitwise MAP decoder. You might expect that to be the same syndrome decoding yes or no think about that think about what that means anyway does not matter but that is that is the different between hard decision and soft decision you are not using the small l s as real numbers.

You are quantizing it to just one bit you are saying either is greater than 0 or less than 0 that is the only thing I will use, I would not use anything else is what you are saying, is there a question luckily that does not change any of the statements. I made about the properties of their expression which are important, so I guess thanks for pointing it out why I cannot choose all 0s. You can do the exact same thing no absolutely nothing changes if you do that, so it depends on the code, so all those things are out there you know i mean so you cannot really make a general statement based on which is the optimal thing to divide by that.

So, that is difficult particularly when your numbers are large like one thousand five hundred and all it is just you can give up. It is not possible to solve those problems, so what I am going to do next real quick is to write a general expression for a general code n k code, so this will it will mirror this expression very closely.

(Refer Slide Time: 58:21)



(n,k) Code C:

$C|_{c_i=0} = \{\underline{c} \in C : c_i = 0\}$

$\downarrow$ Subcode $\longrightarrow$ dim $= k-1$ (mild assumption on C)

$\Downarrow$

$C|_{c_i=1} = \{\underline{c} \in C : c_i = 1\}$, size $= 2^{k-1}$

$$L_i = \log \frac{\sum_{\underline{u} \in C, u_i=0} \prod_{j: u_j=0} e^{l_j}}{\sum_{\underline{u} \in C, u_i=1} \prod_{j: u_j=0} e^{l_j}} \qquad \left(l_j = \frac{2r_j}{\sigma^2}\right)$$

So, you will see how it works it is not so difficult to write down, so if you have an general n k code, so one of the properties you can show is this following thing. If I divide c i equal 0, I am sorry c I am saying conditional on c i equals 0 as the set of all code words in c such that c i is 0. One can show very quickly that this will be a sub code, so this will be a sub space and you can also show it will be a sub space and you can also show that I have only two possibilities for each dimension.

One is k itself, the other is k minus 1, so think about what it means if it is k itself what does it mean the entire code has i th bit equal to 0 which is a really bad thing to do mean you should do that. So, if you assume that the code is such that no one position remains 0 all the time then the dimension of this guy will be equal to k minus 1. You can prove it is not very hard to prove, think about how you made one to prove it there is a problem like this in the tutorial also which has a solution. So, you can check that so the dimension of this sub code equals k minus one if you assume under very some mild assumptions.

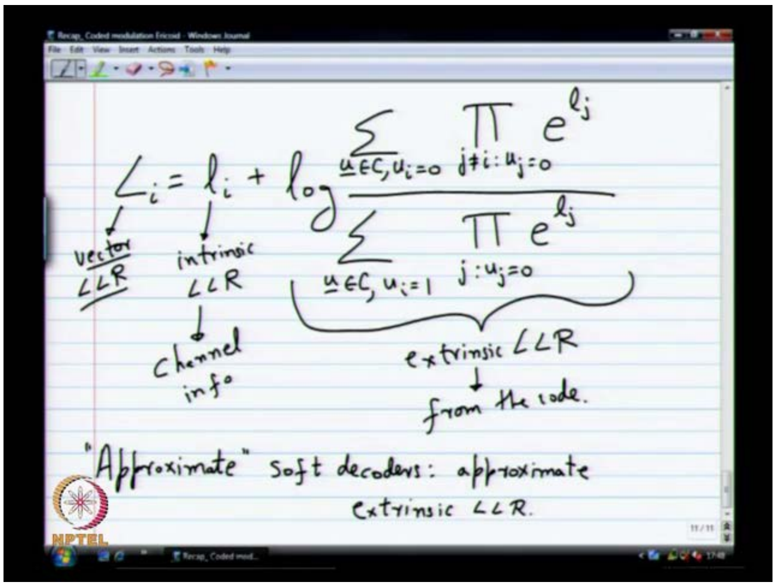
So, what is the mild assumption on code and c the mild assumption is basically that c is such that no one coordinates remains 0 all the time. So, if that is the case the dimension becomes k minus 1, so why this is useful is why this is useful in the bitwise MAP decoder expression the numerator is controlled by code words of this form for capital l because set of all code words with c i equals 0. This tells you that it is exactly 2 power k minus 1, so what about the remaining guys it will be in the denominator. So, those guys in the denominator will also be, so this implies c such that c i is 1 which is the set of all c belonging to c such that c i is 1.

This will be a sub code not be a sub space why will all this why will this not be a sub space yes all 0 is not there. So, it is one way of saying it or you take two code words which have i th bits one you add them what happens i th bit becomes 0 clearly that is not in this set it is not a sub code, but has size equals 2 power k minus 1. So, that follows from the previous result, so this is true in general for any code most linear code will satisfy this construction it does not matter whether i th i is the systematic position or parity position or anything.

So, for any position, this is true as long as you make sure this mild assumption on c s value, so once you have this you can very easily write down L i. So, it will be log what should i write down log there will be a summation over u in c such that u i is 0 or u in c c i is 0 whatever maybe the same thing you writing. Then, what should I write here, so I will write it slightly differently product. I will say j such that u i is 0, e power l j is that divided by you will get a similar expression.

Here, you will have to put summation u in c u i is one product over j such that i am sorry this is u j no u j equals 0 e power l j is that what is small l j l j is 2 r j by sigma square is the scalar l l r. This is the vector l l r s, so that is the generalization of the previous one, so once again you can also do the general intrinsic extrinsic kind of thing.

(Refer Slide Time: 01:03:07)



$$\underset{\downarrow \\ \text{vector LLR}}{L_i} = \underset{\downarrow \\ \text{intrinsic LLR} \\ \downarrow \\ \text{channel info}}{l_i} + \underbrace{\log \frac{\sum\limits_{\underline{u} \in C, u_i = 0} \prod\limits_{j \neq i : u_j = 0} e^{l_j}}{\sum\limits_{\underline{u} \in C, u_i = 1} \prod\limits_{j : u_j = 0} e^{l_j}}}_{\substack{\text{extrinsic LLR} \\ \downarrow \\ \text{from the code.}}}$$

"Approximate" soft decoders: approximate extrinsic LLR.

So, you can show that this will be the same as l i plus log summation u in c such that u i is 0 and product j not equal to i such that u j is 0 e power l j divided by summation. You will see u i is one product j u j is 0 e power l j is that, so here I have just pull out the l i, so if you will previously if j equals i clearly u i will be 0. So, e power l i will show up in each of the terms in the numerator, so I can pull it out and write it in this form, so this is a very general form this is called intrinsic l l r. So, this guy is called extrinsic l l r, so this is the final vector l l r, so this is what channel is also called channel information.

This is the Billy for the probability or the l l r that the channel is giving you only the channel I am telling you about the i th bit what is this this is from the code, basically not just a code from the rest of the received rest of the bits in code. So, basically it brings out how what c i will make the rest of the code behave in a certain way, so that is what that brings out so that is extrinsic to the i th received value. This is intrinsic to the i th received value, so I am combining what come from the channel using the knowledge of your code and bringing it in and you getting the answer.

So, that is a decision you can make when you can, so I have to answer that question very carefully because if you are implementing the entire optimal decoder and complexity is not an issue to you 2 power k minus 1 and k is 500. Then, it is a big issue, but assuming the complexity is not an issue it is enough if you do it only for the message bit, but what people do in practice is like I said they do not evaluate all the terms in the numerator. All the terms in the denominator there is no hope of being able to evaluate all these terms, so you will start with some iterative process you will iterate on it. You will find some terms and then you will further iterate fine further iterate further iterate etcetera when you do that.

You will have to compute for everything that is the difference usually the way it will work is see the c r c will be outside c r c will be on a outer block, the error correction will be inside, so your message itself will have a c r c check. So, if the coding coded codec or the decoder is giving you a message bit that will all not be independent. So, that is the assumption we make in when we design decoders, but obviously that will be part of some IP packet it will have header. It will have whatever is has will be there and there will some dependencies, so it is assuming its independent and all is an assumption, but usually works in practice it is not a bad assumption, but you should be aware that c r c might be there already.

So, if at all CRC is there, usually it will be inside the message you would not put it on, I do not know if it is a good idea to put it on the entire block. If you if you do that, then you have to decode everything, so these are all very difficult to say this kind of answers the question was which will dominate small l i or the other one. So, you have to think about this very carefully, this is depends on how bad your channel is suppose your SNR is very high. Then, what is going to happen L i is going to dominate, so this if at all it goes into error it will be only very minor in error and can be corrected. So, it is what you expect, but when your SNR is bad or average, then small l I also will not be very good, so you are really relying on everything to correct.

In general, as long as your SNR is decent you will get good l l rs you know, but there will be there will be some errors and you have to correct it. So, to answer your question is if the i th bit is received correctly small l i is probably going to good the i th bit is received wrongly. When the extrinsic is the only thing you are hoping for it and it will be usually good if your code is good and if it is if suppose to work at that SNR.

The extrinsic will be good that is the way you have to think about the problem any other question, the thing I want to point out is this structure for the vector l l r or the final l l r is very crucial. So, it is kind of exploited in approximate computations, so you try to compute an intrinsic l l r accurately there is no big deal in computing that. So, it is just 2 r by sigma square anybody can do it, but the extrinsic clearly you cannot compute accurately. So, particularly when k is 500 and all that it is very difficult, but one thing you have to pay attention to is for different i the extrinsic l l r computations will all be related.

So, it is not like you have do the same computations over and over again, you can optimize it little bit, but even if you do that it is very hard it is difficult if k is 500 becomes 2 power 499. It is a very large number, so what people do is they try and approximate the extrinsic they see how good how well you can approximate and practice and your code has to suitable for that is you r code such that approximating the extrinsic is easy. So, in all approximate soft decision decoders the main trick is to approximate the extrinsic l l r in fact the surprising part is the in most practical implementation of these decoders you will erroneously compute the extrinsic l l r.

You will know that you will make you are computing something which is not the extrinsic l l r you can find out that find out find the doubt very easily, but you still go ahead and do it because it turns out even if you erroneously compute it. It is good, it works out very well as long as that the code is not really bad, it works out quite well for the kind of error rates that we want. So, if you do not want like 10 power minus 40 block error rate if you want like 10 power minus 6 which is which is quite reasonable this kind of approximations and erroneous computations of extrinsic l l rs is still good enough. It cannot be totally wrong, you know I cannot just compute the wall that you want there should be some sense in which the error is small.

You know should be able to say that if your block length becomes very large, the error goes to 0 or something you are able to you should able to show that the error will go to zero at some under some conditions. If you cannot do that even if you do an erroneous computation the extrinsic l l r is good enough and you get good answers that is the heart of many of these approximations. So, you compute the extrinsic and you know you will make errors but you bound the errors say the error is in fact you do not even bound the errors you simply say that the error is going to go to 0, if something becomes large or some under some conditions and that becomes good enough.

So, I think I mean I can do more examples, but I think these examples 6, 3 example was good enough. So, this kind of example is good enough, you can go and try the other bits if you want you will get similar answers. It is not too difficult to see that and one more thing I should point out is the bitwise MAP decoder and the m l decoder clearly does not agree here also what you do in the m l decoder for deciding the fourth bit. So, it is kind of similar look at what is inside the exponent, so that is kind of what you do in the m l decoder, so you think go and think about what it means.

You know you need to do correlations with all ones and all minus 1 s, it is roughly this is there in the numerator it is not you do not take the exponentiation and all that. So, one thing in the repetition code we saw that the m l decoder and bitwise MAP are same, so for this 6, 3 code clearly it would not be the same. It will be very different. So, I think we are running out of time, so we will stop here, so what we have seen is optimal decoders. Then, what will see in the next week we do not have class, so week after that the following lectures what we will see is we will see ideas such as capacity what is the best coding gain.

You can do notation of coding gain is something else will introduce in the next lecture and that is very crucial in today's communication systems. Today, you do error control coding or coding for coding gain not really for correcting errors that is the way to think about the communication systems today. So, will stop here error correcting capability those things are very hard to prove, so let me point one thing, so the error correcting capability will not really increase because of this you can only correct so many errors without any ambiguity.

So, that that is definitely true and many of these decoders ultimately their performance is bounded by minimum distance only when you go to really low block error rates. If you are not going to such low block error rates like 10 power minus 40 willing to settle for 10 power minus 6, 10 power 7, 10 power minus 8, then it turns out minimum distance does not control it too much.

Thanks.