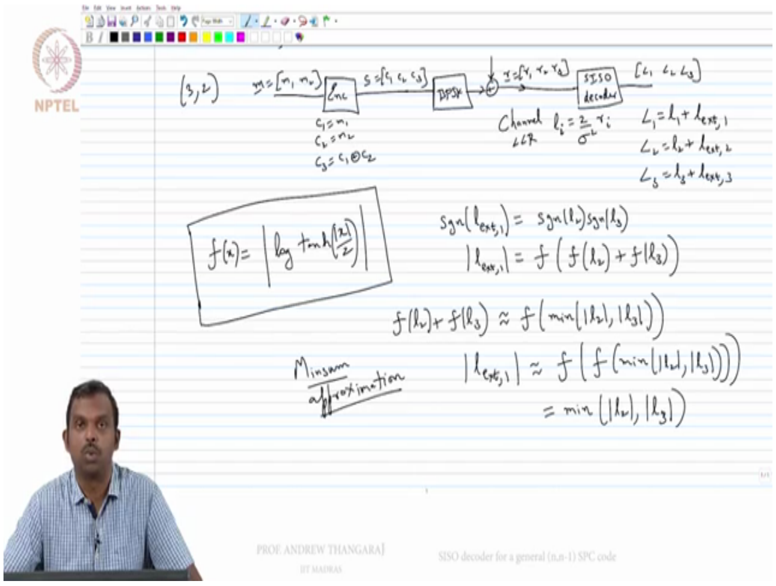
LDPC and Polar codes in 5G Standard
Professor Andrew Thangaraj
Department of Electrical Engineering
Indian Institute of Technology, Madras
SISO decoder for a general (n,n-1) SPC code

(Refer Slide Time: 00:16)



$$c_1 = m_1,\quad c_2 = m_2,\quad c_3 = c_1 \oplus c_2$$

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

$$L_1 = l_1 + l_{ext,1},\quad L_2 = l_2 + l_{ext,2},\quad L_3 = l_3 + l_{ext,3}$$

$$\boxed{f(x) = \left|\log \tanh\left(\frac{|x|}{2}\right)\right|}$$

$$\text{sgn}(l_{ext,1}) = \text{sgn}(l_2)\,\text{sgn}(l_3)$$

$$|l_{ext,1}| = f\big(f(l_2) + f(l_3)\big)$$

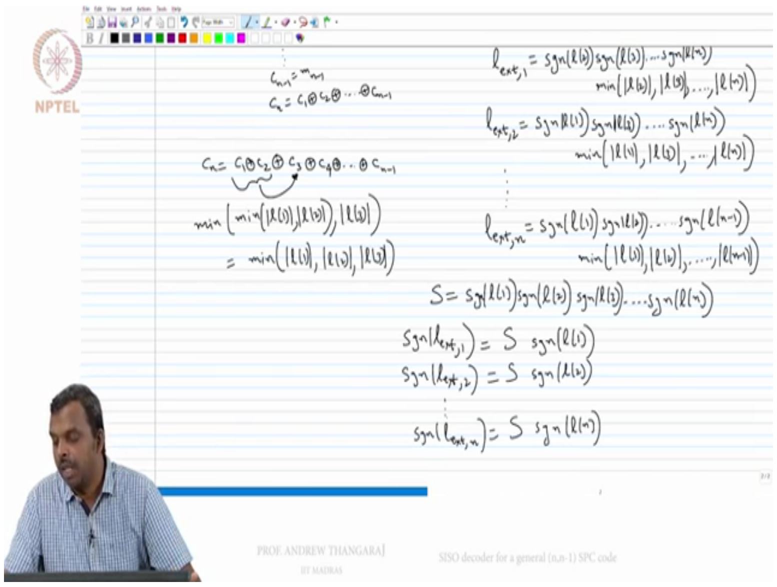
$$f(l_2) + f(l_3) \approx f\big(\min(|l_2|, |l_3|)\big)$$

Minsum approximation

$$|l_{ext,1}| \approx f\Big(f\big(\min(|l_2|, |l_3|)\big)\Big) = \min(|l_2|, |l_3|)$$

Hello, in the previous lecture we saw the single parity check decoder for the 3 comma 2 case and we saw this min sum approximation and we saw an illustration of how these things happen for some simple examples and particular scenarios. So in this lecture we will extend this idea to longer codes instead of 3 comma 2, if you have n comma n minus 1, ok then what do you do? How do you do this? So it turns out it is quite easy to extend but I want to describe that and then in this lecture, ok.

(Refer Slide Time: 00:48)



$c_{n-1} = m_{n-1}$

$c_n = c_1 \oplus c_2 \oplus \cdots \oplus c_{n-1}$

$c_n = \underbrace{c_1 \oplus c_2}_{} \oplus c_3 \oplus c_4 \oplus \cdots \oplus c_{n-1}$

$\min\left(\min(|l(1)|, |l(2)|), |l(3)|\right)$

$= \min(|l(1)|, |l(2)|, |l(3)|)$

$l_{ext,1} = \text{sgn}(l(2))\,\text{sgn}(l(3)) \cdots \text{sgn}(l(n))$
$\min(|l(2)|, |l(3)|, \ldots, |l(n)|)$

$l_{ext,2} = \text{sgn}(l(1))\,\text{sgn}(l(3)) \cdots \text{sgn}(l(n))$
$\min(|l(1)|, |l(3)|, \ldots, |l(n)|)$

$\vdots$

$l_{ext,n} = \text{sgn}(l(1))\,\text{sgn}(l(2)) \cdots \text{sgn}(l(n-1))$
$\min(|l(1)|, |l(2)|, \ldots, |l(n-1)|)$

$S = \text{sgn}(l(1))\,\text{sgn}(l(2))\,\text{sgn}(l(3)) \cdots \text{sgn}(l(n))$

$\text{sgn}(l_{ext,1}) = S\ \text{sgn}(l(1))$

$\text{sgn}(l_{ext,2}) = S\ \text{sgn}(l(2))$

$\vdots$

$\text{sgn}(l_{ext,n}) = S\ \text{sgn}(l(n))$

PROF. ANDREW THANGARAJ
IIT MADRAS

SISO decoder for a general (n,n-1) SPC code

So the picture is similar to before except that we are using n comma n minus 1, so you have a message bit message vector which is m 1, m 2, till m n minus 1 then you do the single parity check code which will give you a code word c which if I say c 1, c 2 till c n we know how this is done c 1 equals m 1, c 2 equals m 2 dot till c n minus 1 equals m n minus 1 and then c n becomes c 1 XOR, c 2 XOR, c n minus 1, so this is the single parity check code and then you do BPSK then you get received values it is noise get received values r 1 r 2 to r n and then you have a SIDO decode and you put out capital L which is L 1, L 2, L n this is also l plus l x, ok and what is l? l is 2 by Sigma square times r and this l x is the critical calculation, ok.

So how do we calculate this? We know how to do this, l x 1 is basically sign of l of two times sign of. So we saw for the 3 comma 2 case for the extrinsic value for the first bit you find the product of the signs of L of 2 and L of 3 and then you take the minimum of the absolute value of L of 2 and L of 3, ok. So it turns out the same thing extends for a longer single parity check code, ok. So I will first provide what the answer is and then go into details of how that comes up, ok.

So what is a l x 1? So this is simply the product of the signs of all the remaining channel LLR is and then minimum of absolute value of all the I am sorry l of n, ok. So just like we had for 3 comma 2 for the first extrinsic, we took all the remaining values multiplied their signs and took the min of the absolute values, so likewise when you have n comma n minus 1 for the extrinsic value for the first received first bit you simply take the remaining LLR is multiply them and then take the minimum of the absolute values for that also, ok.

So this is l x 1, so what is l x 2? so it does not vary too much from this, you take sign of l of 1, sign of l of 3 you drop the first second l of 2 then after that it is all going all the way till the end and then the absolute value it is min of absolute value of l of 1, absolute value of l of 3, so on till absolute value of l of n so, ok. So that is what you do for the second extrinsic value, ok.

So likewise you can go on what happens for the last extrinsic value you will only take the first n minus 1 and then minimum for the first n minus 1, ok. So this is quite a reasonable extension so because if you see when you do c n you have c 1 XOR, c 2 XOR, so on till maybe I should write a few more terms here ok, so XOR c 3, XOR c 4 so on till XOR c n minus 1, ok.

So first you have c 1 plus c 2, ok. So we know how to deal with c 1 plus c 2 we do minimum of those two LLR is and product of the signs and then you have that XOR with c 3, so if you look at what is going on here min of absolute value of l of 1 comma absolute value of l of 2 and then if you take min of that with absolute value of l of 3 that is the same as the overall minimum, right ok.

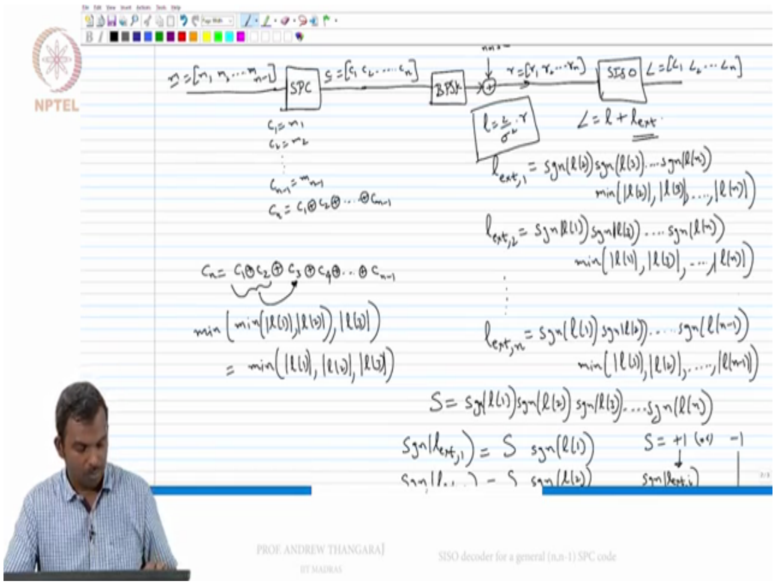
So it is enough if you take the minimum of all of the every other input LLR other than the first other than the nth 1 in this case, ok. Same thing happens with the sign, so you do sign of l of 1 times sign of l of 2 and then multiply that with the sign of l of 3, it is the same thing as multiplying all the signs together, ok. So extending from 3 comma 2 to n comma n minus 1 is sort of immediate as far as the SISO decoder is concerned, ok.

Now it seems here that you are doing a lot of operations ok and it turns out one can simplify this in a very nice way, ok. So this is quite important and this is how people implement the actual decoder. So first is with the sign, ok so what people would do is, you would first find the product of all the signs ok, sign of l of 1, sign of l of 2, sign of l of 3, so on till sign of l of m, ok you find the product of all the signs, ok so that is one number.

So the sign of l x 1 is actually S times sign of l of 1 ok why is this true? Ok. If you take S and multiply with sign of l of 1 what happens you get sign squared of l of 1 and all these other things. What is sign squared of l of 1? Sign square of x is actually 1, ok why is that? The sign is minus 1 sign into sign becomes plus 1, sign is plus 1 everything is plus 1, ok. So sign squared becomes 1, so that goes away.

Now this is actually true for every sign so in fact sign of l x 2 is simply S times sign of l 2 and so on, ok. Sign of l x n is S times sign of l of n, ok. So this is easy enough the sign is taken care of, ok. What about the absolute value? It looks like you have to find so many minimums for every extrinsic you have to find the minimum of n minus 1 things, ok. So it turns out one can rationalize that as well, ok.

(Refer Slide Time: 08:06)



$m = [m_1\ m_2 \cdots m_{n-1}]$
SPC
$c = [c_1\ c_2 \cdots c_n]$
BPSK
$r = [r_1\ r_2 \cdots r_n]$
SISO
$L = [L_1\ L_2 \cdots L_n]$
$l = \frac{2}{\sigma^2} r$
$L = l + l_{ext}$
$c_1 = m_1$
$c_2 = m_2$
$c_{n-1} = m_{n-1}$
$c_n = c_1 \oplus c_2 \oplus \cdots \oplus c_{n-1}$
$l_{ext,1} = \text{sgn}(l(2))\,\text{sgn}(l(3)) \cdots \text{sgn}(l(n))\ \min(|l(2)|, |l(3)|, \ldots, |l(n)|)$
$l_{ext,2} = \text{sgn}(l(1))\,\text{sgn}(l(3)) \cdots \text{sgn}(l(n))\ \min(|l(1)|, |l(3)|, \ldots, |l(n)|)$
$c_n = c_1 \oplus c_2 \oplus c_3 \oplus c_4 \oplus \cdots \oplus c_{n-1}$
$\min(\min(|l(1)|, |l(2)|), |l(3)|) = \min(|l(1)|, |l(2)|, |l(3)|)$
$l_{ext,n} = \text{sgn}(l(1))\,\text{sgn}(l(2)) \cdots \text{sgn}(l(n-1))\ \min(|l(1)|, |l(2)|, \ldots, |l(n-1)|)$
$S = \text{sgn}(l(1))\,\text{sgn}(l(2))\,\text{sgn}(l(3)) \cdots \text{sgn}(l(n))$
$\text{sgn}(l_{ext,1}) = S\ \text{sgn}(l(1))$
$S = +1 \ (+1) \ -1$
$\text{sgn}(l_{ext,i})$
PROF. ANDREW THANGARAJ
IIT MADRAS
SISO decoder for a general (n,n-1) SPC code

So what you have to do for that is, you find first of all m 1 which is the min of absolute value of l of 1, absolute value of l of 2, so on till absolute value of l of n, ok. So this is the overall minimum absolute value this is m 1, ok and then you also find the position of this which is the argument of min of l of 1, ok. What do you mean by position on the argument? So out of these n values, the minimum will occur in some ith position, right so that is this pos where does minimum occur, so this pos could be 1 or 2 or 3 all the way up to n, any one of these values.

Now you might say ok there may be two values which are equal ok, if two values are equal you take any one of those to be pos ok, so it does not matter if both 1 & 2 are the same l of 1 and l of 2 are the same you can take any one of them to be this value of pos. So I have the overall minimum m 1 and then the position where that overall minimum occurred ok and then I also need the overall second minimum, ok.

So what is this? This is m 2 let me call it, this is min of so I will write it like this so it should be clear to you hopefully, so l of 1 l of 2 all the way till l of pos minus 1 ok and then l of pos plus 1 till l of n, ok. So this is the overall second minimum, so what is going on here I find the overall minimum and then find out where that minimum occurs and then I knock out that value, ok.

So I take that l of pos and throw it out remember l of pos is m1 equals l of pos right, so that is the point here where does the minimum occur. So I find where the first minimum occurs and throw it out and among the remaining values find the minimum again, so this is the overall second minimum and that is m, m 2, ok, is it ok. So this is the idea so you take all the incoming LLR is, take their absolute values, find the overall first minimum where it occurs then throw it out find the overall second minimum, ok.

And once you do this it turns out this absolute value of l x pos equals m 2 and then absolute value of l x i for i not equal to pos is m 1, ok so this is the main result that you can have, ok. So think about why this is true? So what is the absolute value is, absolute value of l x is basically the minimum absolute value of all the remaining S, ok. Now there is an overall minimum and an overall second minimum for the position of the overall minimum the absolute value extrinsic absolute value is actually the overall second minimum and for every other position the absolute value of the extrinsic LLR is the overall first minimum, ok so that's the idea and you can think about why this is true but it will work out, ok.

So one can find the absolute values very easily, one can find the signs very easily in fact what is going on here is, so S could be either plus 1 or minus 1 right ok, if it is plus 1 so you see sign of l x i is equal to sign of l i right, if it is minus 1 sign of l x i is minus sign of l of I, ok. So this is the interesting part here S gets multiplied by the sign, so once you find S, once you find the absolute value a, you either flip all the signs or you do not flip all the signs, ok.

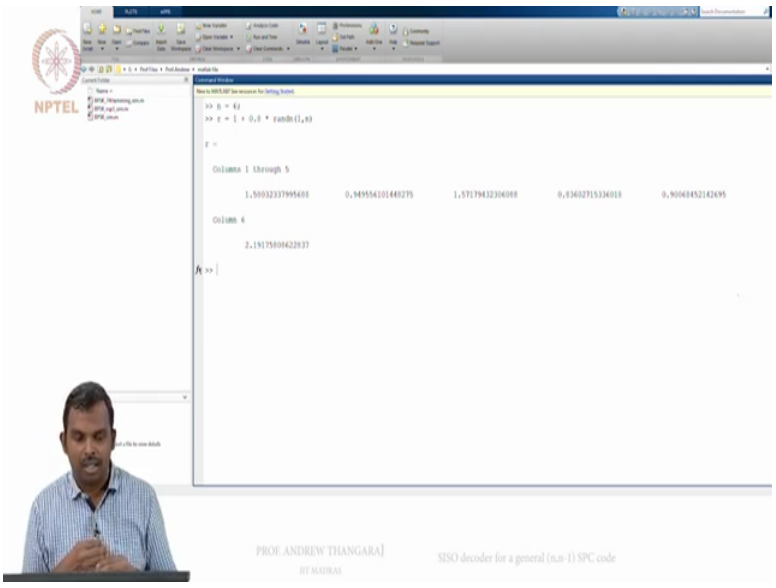
If the S is negative, S is minus 1 you flip the signs otherwise you do not flip the signs ok, so this is the basic operation for the SISO decoder for the single parity check code, ok. Hopefully this is clear to you once again let me repeat, when you have the n comma n minus 1 single parity check code how do you do the SISO decoder? You compute the intrinsic LLR is first this is the channel or intrinsic LLR 2 by Sigma square times r and then from that you compute the extrinsic LLR.

Now how do you find the extrinsic LLR, we will use them in some approximation we are not going to do anything else first you find the product of the signs of all the LLR is, ok that is your S, this could be either plus 1 or minus 1 and then you find the absolute value of the extrinsic information. How do you find the absolute value of the extrinsic information? You find the overall minimum and it is position and the overall second minimum.

The x absolute value for the overall minimum position pos is actually m 2 the overall second minimum and for all other values the absolute value is the overall minimum m 1, ok. That is the important thing to realize here and then how do you do the sign? If the overall product of the signs was plus 1 you retain the signs as it is same as that of the channel LLR but in case the S is minus 1 you flip all the sides, ok.

So what I am going to do next is take some sample noise values and illustrate how this decoding happens, ok. So it is quite a quite important that we see a few of these decoders and how they work, how these values occur and so on, ok and we will stick to this simple mean some approximation, ok.

(Refer Slide Time: 14:18)

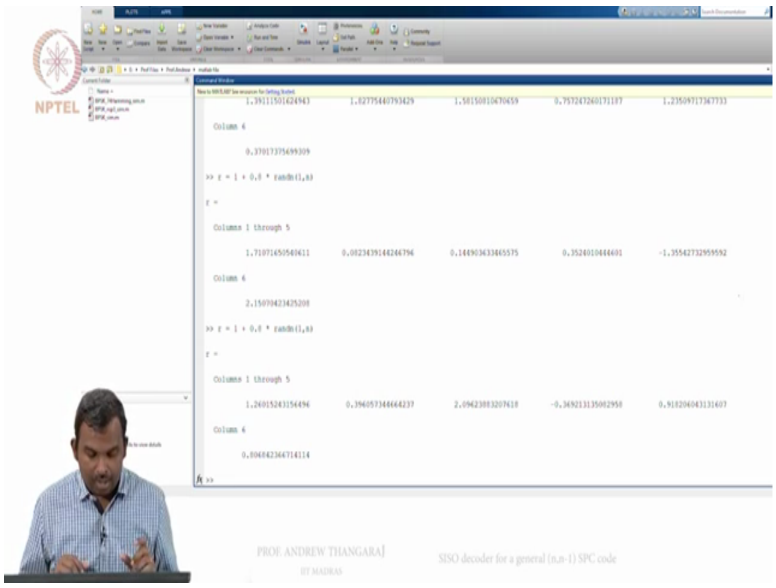


```
>> n = 6;
>> r = 1 + 0.8 * randn(1,n)

r =

  Columns 1 through 5

        1.50032337995600        0.949556101440275        1.57179432306080        0.83602715336010        0.90068452142695

  Column 6

        2.19175800622037

fx >>
```

So let us let me clear this up, ok. So I am going to let the receive value, so I am going to take all 1 is, ok. So as my code word so what am I doing here so I am fixing my message to be all 0, ok. So this is actually a usual thing to illustrate and you can always fix the message to be all 0, you can take something else as well but usually all 0 is good enough, ok. I am fixing my message to be all 0 which means the code word is also all 0 ok and the symbol vector is always plus 1 is but of course the noise will vary, so you will get them, ok.

So what about n let us take n to be number 6, seems like a nice number to take and then r s 1 plus so I need a Sigma value, so I will stick to the same values before point 8 random of 1 comma n, so this is the received value, ok. So you get all these values 1 point 58 point 94 there is no noise in this one.

(Refer Slide Time: 15:15)

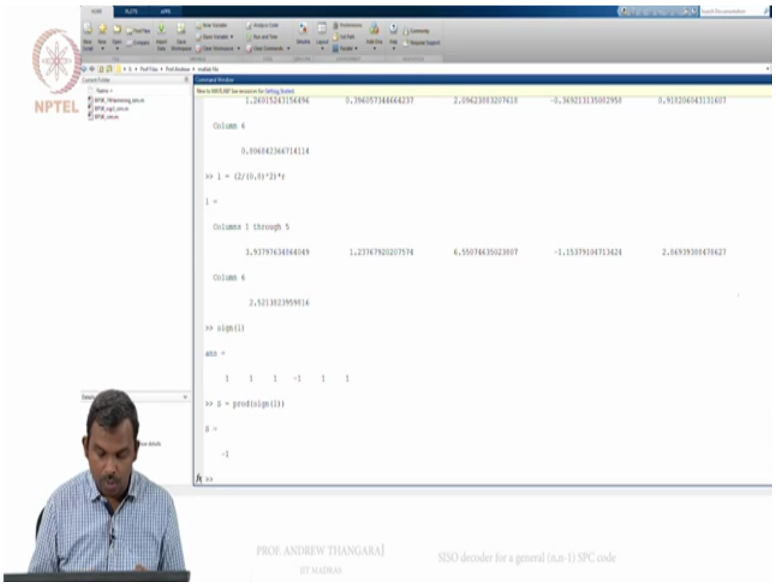


```
        1.39111501624943      1.02775440793429      1.50150010470659      0.757247260171107      1.23509717367733

  Column 6

        0.37017375699309

>> r = 1 + 0.8 * randn(1,n)

r =

  Columns 1 through 5

        1.71071650540611      0.0023439144246796      0.144903633465575      0.3524010444601      -1.35542732959592

  Column 6

        2.15070423425208

>> r = 1 + 0.8 * randn(1,n)

r =

  Columns 1 through 5

        1.26015243156496      0.396057344664237      2.09623083207610      -0.369213135002958      0.910206043131607

  Column 6

        0.806842366714114

fx >>
```

Maybe I can try one more, okay maybe I can try one more, this noise but this no it is not there is no error in the hard decision decoding ok there you go, you got an error, it is a pretty big error in some sense let us see what happens here, ok.

So that is the one of the r is so, ok so this one looks a bit more reasonable, so let us take this to be the receipt vector, ok. So you see different values of r will give you different illustrations of the decoders I am just trying to find a good value of r, so that good values of the received vector, so that we have a good illustration. So let us look at this received value r here 1 point 26, point 39, 2 point 09 minus point 36, point 91, point 80, ok. So there is an error, so something will happen, ok.

(Refer Slide Time: 16:10)

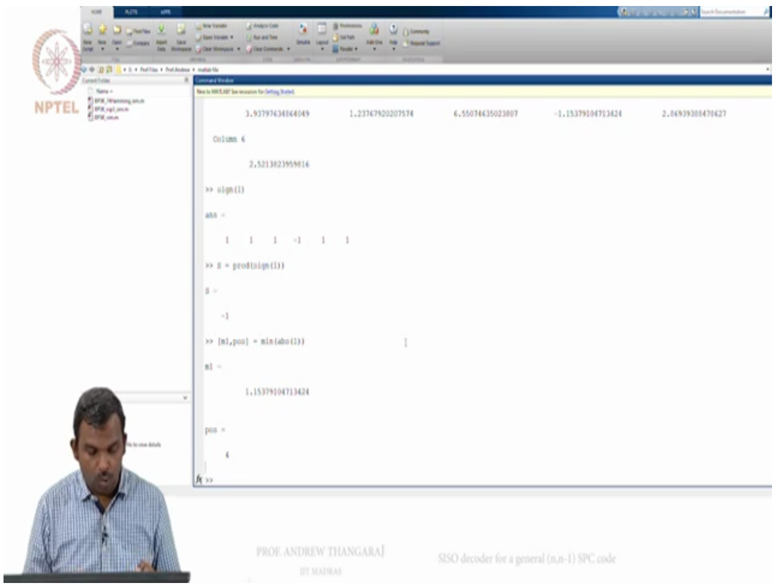


```
   1.26015243156496    0.396057344664237    2.09423083207418    -0.369213135002958    0.918206043131607

  Column 6

   0.806842366714114

>> l = (2/(0.8)^2)*r

l =

  Columns 1 through 5

   3.93797634064049    1.23767920207574    6.55074635023007    -1.15379104713424    2.86939300470627

  Column 6

   2.5213823959816

>> sign(l)

ans =

     1     1     1    -1     1     1

>> S = prod(sign(l))

S =

    -1

fx >>
```

So the first is to calculate l, l as you know is 2 by Sigma squared times r, ok so we get this ok. 13 point 93, 1 point 23, 6 point 55, so on ok and then what do we do we need to do l x for l x I need the product of the signs, right. So if you do sign of l, you get the signs 111 minus 111 and then you can write down capital S as the product of sign of l ok that is minus 1, ok.

I could have also told you that but still just to illustrate we do this, ok and then we now have to find the absolute values, right. So absolute value and the minimum absolute value is that right, so mat lab has a lot of functions for this kind of a thing.

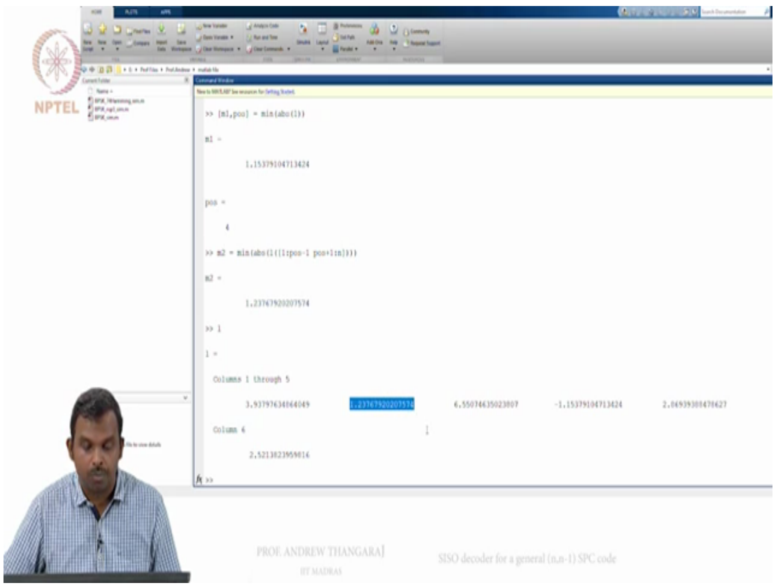
(Refer Slide Time: 17:07)



```
   3.93797634064049    1.23767920207574    6.55074635023007    -1.15379104713424    2.86939300470627

  Column 6

   2.5213823959816

>> sign(l)

ans =

     1     1     1    -1     1     1

>> S = prod(sign(l))

S =

    -1

>> [ml,pos] = min(abs(l))

ml =

   1.15379104713424


pos =

     4

fx >>
```

So you can do for instance m 1 comma pos equals min of abs of L. So this tells you that the minimum value occurred at the fourth position 1, 2, 3, 4 and the fourth position is 1 point 1537 that is pretty decent, you see that that is the minimum value and it is the fourth position, ok.

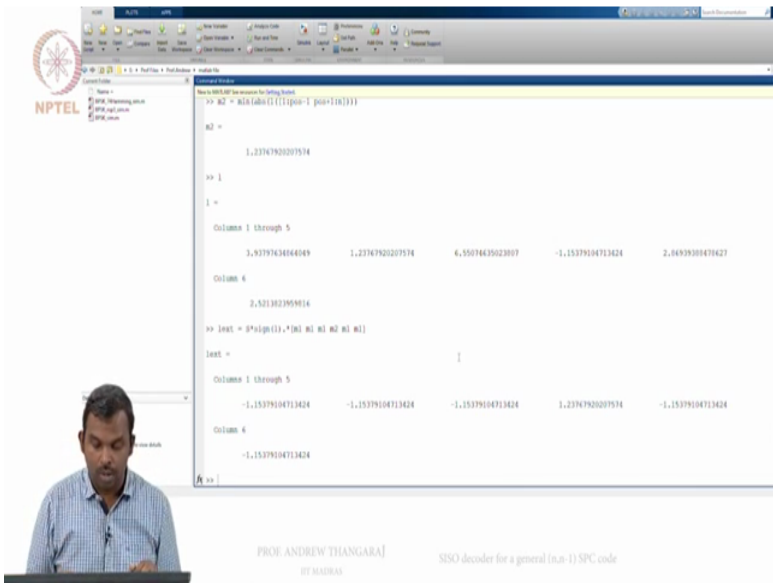
(Refer Slide Time: 17:31)



```
>> [m1,pos] = min(abs(l))

m1 =

        1.15379104713424


pos =

     4

>> m2 = min(abs(l([1:pos-1 pos+1:n])))

m2 =

        1.23767920207574

>> l

l =

  Columns 1 through 5

        3.93797634064049        1.23767920207574        6.55074635023007        -1.15379104713424        2.06939301470627

  Column 6

        2.52138239590l6

>>
```

So how do I find m 2? m 2 is mean of abs of l but l I want one colon pos minus one and then, so I need to this put it into two brackets pos plus one colon n, is it. So minimum of everything else other than the first minimum, so that is m 2, let us check once again that is correct, so you see m 1 is the first minimum, it is this 1 point 15379 in the fourth position, m 2 is the second minimum and that is 1 point 23, ok.

(Refer Slide Time: 18:14)



```
>> m2 = min(abs(l([1:pos-1 pos+1:n])))

m2 =

        1.23767920207574

>> l

l =

  Columns 1 through 5

        3.93797634064049        1.23767920207574        6.55074635023007       -1.15379104713424        2.06939300470627

  Column 6

        2.5213023959016

>> lext = S*sign(l).*[m1 m1 m1 m2 m1 m1]

lext =

  Columns 1 through 5

       -1.15379104713424       -1.15379104713424       -1.15379104713424        1.23767920207574       -1.15379104713424

  Column 6

       -1.15379104713424

fx >>
```

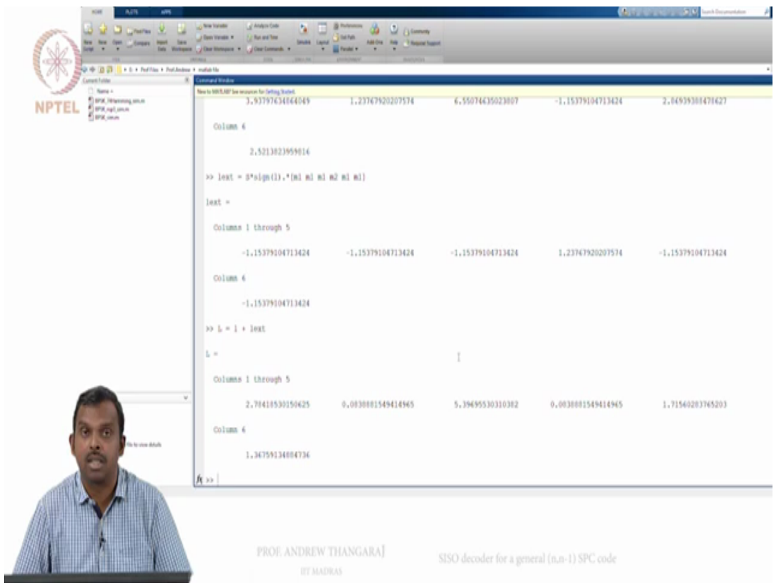
So what do I do for l x, so remember l x just mostly going to be m 1, so you can let l m x l x be simply m 1, so how do I do m 1 here it is just m 1, m1, m 1 except for the fourth position, ok mostly going to be m 1 except for the fourth position where I have to put m 2 ok and then m 1, m 1 is that, ok so that is the absolute value of l x, ok.

So this is the absolute value of l x but I also need signs right, so how do i do signs i have this overall sign S ok and then i also have to multiply with the sign of l, right so I can take sign of l dot into this guy, ok. So what is going on here once again I am finding multiplying S with the sign of l ok, so that gives me the individual signs and then dot into each sign is getting multiplied by the absolute value, I put m 1, m 1, m 1 for the first three guys and wherever the first minimum occurs I need him to and then m 1, m 1 throughout, so this is my l x, so that is how it looks, here ok.

So the extrinsic LLR will look like this, most of the values will all be equal and then at least the minimum first minimum location alone will be different the second minimum will come and the signs will get flipped or not depending on whether the overall parity is satisfied whether the overall product of the signs is positive or not, if the overall product of the signs is negative then all the signs get flipped otherwise none of the signs gets flipped, ok.

So once again look at this how this calculation is gone, we found the overall sign and then the sign of each input LLR and then flipped all the signs and then the absolute value is found by the minimum absolute value minimum two absolute values, most of the values get replaced by the least of the absolute values and then the position of the least alone gets replaced by the second least absolute value, is it ok. So this is the mean some approximation.

(Refer Slide Time: 20:21)



```
      3.93797634844049        1.23767920207574        6.55074435023007       -1.15379104713424        2.04939300470627

  Column 6

      2.5213023959016

>> lext = S*sign(l).*[m1 m1 m1 m2 m1 m1]

lext =

  Columns 1 through 5

     -1.15379104713424       -1.15379104713424       -1.15379104713424        1.23767920207574       -1.15379104713424

  Column 6

     -1.15379104713424

>> L = l + lext

L =

  Columns 1 through 5

      2.78418530150625        0.0830881549414965       5.39695530310382        0.0830881549414965       1.71560203765203

  Column 6

      1.36759134804736

fx >>
```

So if you want to do capital L, you can do L plus L x and you see that is the overall output LLR and this is corrected here, it is ok. So that was a illustration of how the single parity check code works for a larger block length and hopefully you can see this and repeat it for other cases, ok. So this is the end of this lecture to summarize we have seen how to do SISO decoding for the single parity check code and we see that it is easy, it is quite easy to implement even for long block lengths, it just involves finding minimums and product of signs etcetera, it is not very expensive at all in terms of computation.

Now the next important thing is how is this going to be an ingredient in a decoder for a larger code, ok so like I said single parity check code is not very good in correcting errors it can correct no errors in fact and maybe in some cases it can correct errors but in many cases it cannot correct errors, the minimum distance is not good enough, minimum distance is very small, so in reality people use longer codes with lower rate, ok.

So for instance rate half if you look at it you might have a 2000 comma 1000 code, it might be a very large code and how do you use a single parity check as a ingredient in a building a decoder for a larger code, ok so in particular a low density parity check code, ok so how this decoder is used as an ingredient in decoding LDPC codes, is the next topic and we will take it up in the next lecture, thank you very much.