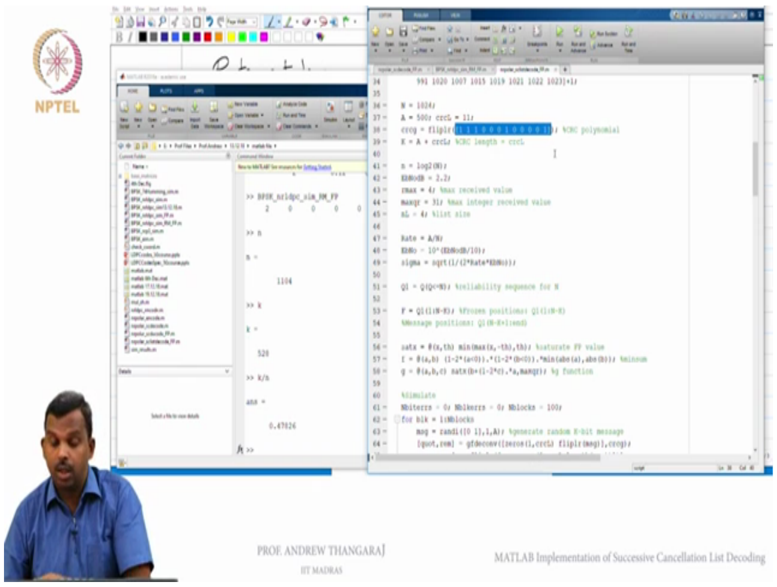
LDPC and Polar codes in 5G Standard
Professor Andrew Thangaraj
Department of Electrical Engineering
Indian Institute of Technology, Madras
MATLAB Implementation of Successive Cancellation List Decoding

(Refer Slide Time: 0:16)



```
>> BPSK_nrldpc_sim_RM_FP
     2     0     0     0     0

>> n

n =

        1104

>> k

k =

   528

>> k/n

ans =

    0.47826
```

```
        991 1020 1007 1015 1019 1021 1022 1023]+1;

    N = 1024;
    A = 500; crcL = 11;
    crcg = fliplr([1 1 1 0 0 0 1 0 0 0 0 1]); %CRC polynomial
    K = A + crcL; %CRC length = crcL

    n = log2(N);
    EbNodB = 2.2;
    rmax = 4; %max received value
    maxqr = 31; %max integer received value
    nL = 4; %list size

    Rate = A/N;
    EbNo = 10^(EbNodB/10);
    sigma = sqrt(1/(2*Rate*EbNo));

    Q1 = Q(Q<=N); %reliability sequence for N

    F = Q1(1:N-K); %Frozen positions: Q1(1:N-K)
    %Message positions: Q1(N-K+1:end)

    satx = @(x,th) min(max(x,-th),th); %saturate FP value
    f = @(a,b) (1-2*(a<0)).*(1-2*(b<0)).*min(abs(a),abs(b)); %minsum
    g = @(a,b,c) satx(b+(1-2*c).*a,maxqr); %g function

    %Simulate
    Nbiterrs = 0; Nblkerrs = 0; Nblocks = 100;
    for blk = 1:Nblocks
        msg = randi([0 1],1,A); %generate random K-bit message
        [quot,rem] = gfdeconv([zeros(1,crcL) fliplr(msg)],crcg);
```

NPTEL
PROF. ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation List Decoding

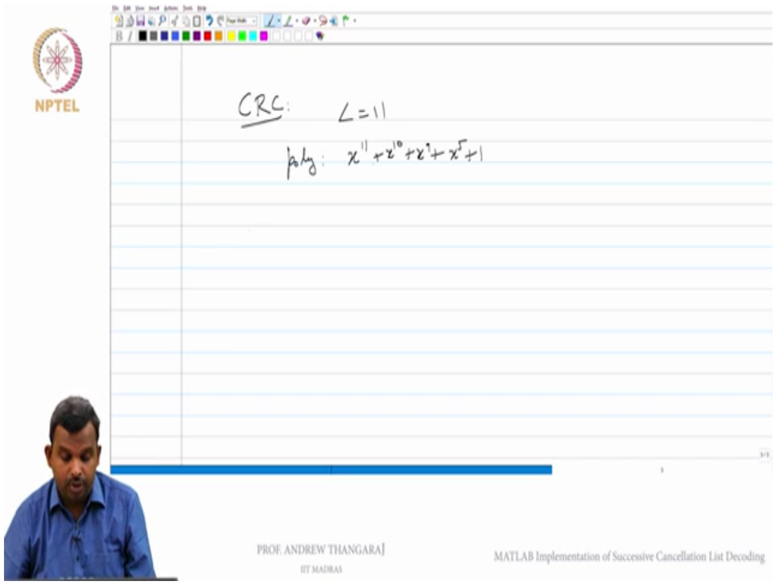
Hello and welcome to this lecture on implementing list decoders for polar codes. So I gave a couple of, I gave one lecture earlier on how to do list decoding, I described the whole philosophy from high level, the path matrix and decision matrix and adding the decision matrix to the path matrix and comparing path matrix and having a list size of 4 picking the least path matrix all of that we described earlier and I also described to you the block diagram of how a list decoder works and how the CRC insertion is important.

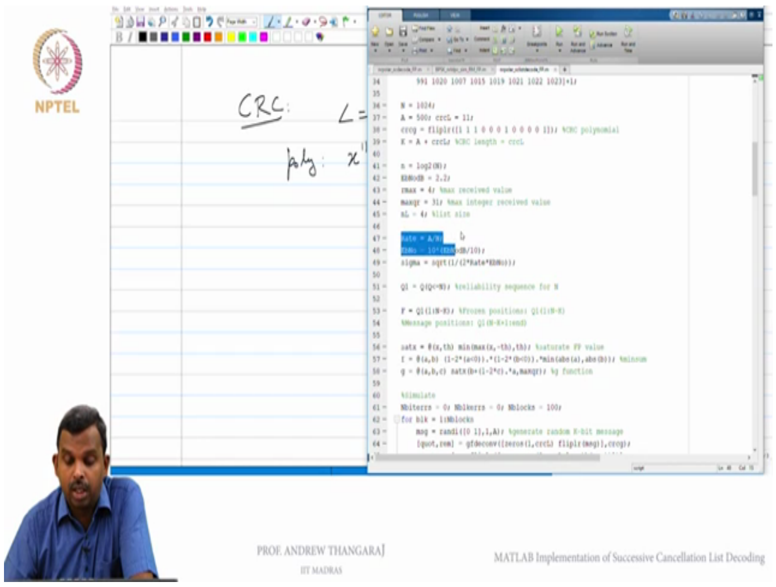
So the list decoder is going to put out the list of code words from that how do you pick the code word that is a valid code word you use the CRC check. So in this lecture we will see how to implement that, we will take our existing success of cancellation decoder and add the list decoding components, you will see it is sort of okay it is easy to do in MATLAB atleast you simply instead of having one decoder you will have multiple decoders and once again we will not be doing this efficiently at all, I will just do a decoder to get it to work and we will think about adding efficiency and other things maybe later on in fact I will leave it to you as the how to improve the efficiency in these decoders.

So now let us just look at how the implementation goes, okay. So the first step like I mentioned is to add CRC without CRC the list decoding is not going to work it does not work sense at all. So you saw in the LDPC code also actually in the standard that there is a CRC but we never implemented it, it is not needed in the LDPC code, but in this case CRC is critical for the list decoder.

So here is where what I have done with the CRC, so you notice quite a few things have changed, I made A as the number of actual message bits and CRCL is the number of CRC bits and K which is the message dimension of the code is A plus CRCL, so 500 plus 11 is 511, okay and for the CRC comes with the polynomial the polynomial is clearly defined in the standard that is the polynomial for you.

(Refer Slide Time: 2:32)


CRC: $L=11$
poly: $x^{11}+x^{10}+x^{9}+x^{5}+1$
NPTEL
PROF. ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation List Decoding


CRC: $L=$
poly: x^{1}
NPTEL
PROF. ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation List Decoding

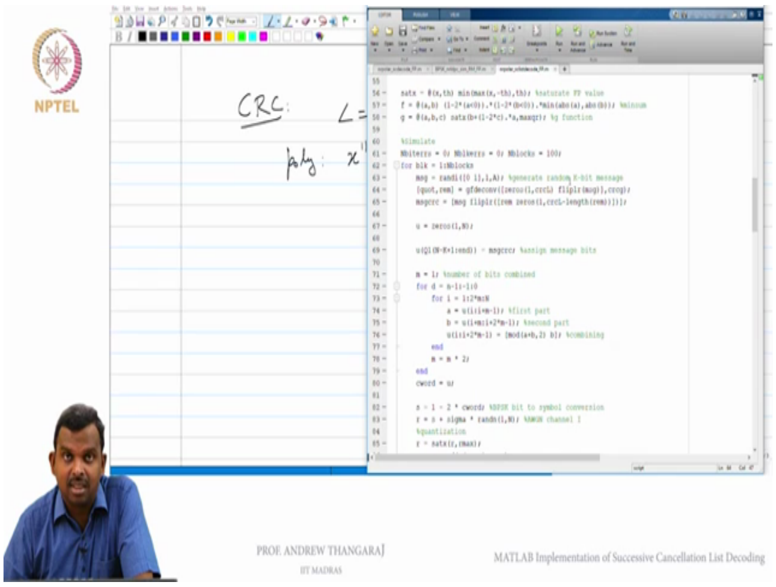
```
    991 1020 1007 1015 1019 1021 1022 1023]+1;

N = 1024;
A = 500; crcL = 11;
crcg = fliplr([1 1 1 0 0 0 1 0 0 0 0 1]); %CRC polynomial
K = A + crcL; %CRC length = crcL

n = log2(N);
EbNodB = 2.2;
rmax = 4; %max received value
maxqr = 31; %max integer received value
nL = 4; %list size

Rate = A/N;
EbNo = 10^(EbNodB/10);
sigma = sqrt(1/(2*Rate*EbNo));

Q1 = Q(Q<=N); %reliability sequence for N

F = Q1(1:N-K); %Frozen positions: Q1(1:N-K)
%Message positions: Q1(N-K+1:end)

satx = @(x,th) min(max(x,-th),th); %saturate FP value
f = @(a,b) (1-2*(a<0)).*(1-2*(b<0)).*min(abs(a),abs(b)); %minsum
g = @(a,b,c) satx(b+(1-2*c).*a,maxqr); %g function

%Simulate
Nbiterrs = 0; Nblkerrs = 0; Nblocks = 100;
for blk = 1:Nblocks
    msg = randi([0 1],1,A); %generate random K-bit message
    [quot,rem] = gfdeconv([zeros(1,crcL) fliplr(msg)],crcg);
```

So if you write it out as a polynomial, okay so the CRC for length equals 11, polynomial is defined in the standard let us write there, okay 1 1 1 0 0 0 1 so this is x power 11 plus x power 10 plus x power 9, 8, 7, 6, x power 5, okay polynomial is x power 11 plus x power 10 plus x power 9 plus x power 5 plus 1, okay so this is the polynomial. And the way in which you do CRC we are not going to describe the ideas of CRC and all that in this class but we will see how to implement it in MATLAB atleast. So this is the CRC polynomial you need this.

So if you choose to do a CRC of a different length let us say length 16 or something then so if you chose to do a polynomial CRC of a longer length say 16 or 24 these are also possible then you have to change your CRC polynomial, we will change it to something else, okay. Alright, so now the next thing I put so you see the quantization all of that is fine, next thing I have is list size, okay so I am picking that as 4, you might want to have 8 maybe later on if you want more performance but 4 is good enough number and seems like a reasonable number.

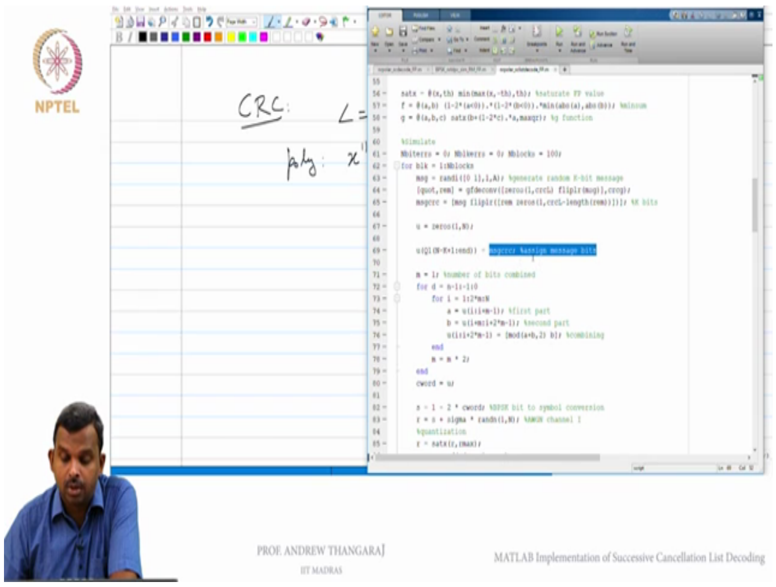
And notice here rate becomes A by N, it is not K by N anymore it is A by N the reason is CRC bits are actually parity bits, they are not message bits and we are using those parity bits in the decoding and the list decoder. So the rate is A by N and that is something important as well, okay so you should capture that as well, okay.

(Refer Slide Time: 4:16)



CRC: L =
poly: x^11
NPTEL
PROF. ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation List Decoding

```
satx = @(x,th) min(max(x,-th),th); %saturate FP value
f = @(a,b) (1-2*(a<0)).*(1-2*(b<0)).*min(abs(a),abs(b)); %minsum
g = @(a,b,c) satx(b+(1-2*c).*a,maxqr); %g function

%Simulate
Nbiterrs = 0; Nblkerrs = 0; Nblocks = 100;
for blk = 1:Nblocks
    msg = randi([0 1],1,A); %generate random K-bit message
    [quot,rem] = gfdeconv([zeros(1,crcL) fliplr(msg)],crcg);
    msgcrc = [msg fliplr([rem zeros(1,crcL-length(rem))])];

    u = zeros(1,N);

    u(Q1(N-K+1:end)) = msgcrc; %assign message bits

    m = 1; %number of bits combined
    for d = n-1:-1:0
        for i = 1:2*m:N
            a = u(i:i+m-1); %first part
            b = u(i+m:i+2*m-1); %second part
            u(i:i+2*m-1) = [mod(a+b,2) b]; %combining
        end
        m = m * 2;
    end
    cword = u;

    s = 1 - 2 * cword; %BPSK bit to symbol conversion
    r = s + sigma * randn(1,N); %AWGN channel I
    %quantization
    r = satx(r,rmax);
```



NPTEL
CRC:
$L =$
poly: x
PROF. ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation List Decoding

```
satx = @(x,th) min(max(x,-th),th); %saturate FP value
f = @(a,b) (1-2*(a<0)).*(1-2*(b<0)).*min(abs(a),abs(b)); %minsum
g = @(a,b,c) satx(b+(1-2*c).*a,maxqr); %g function

%Simulate
Nbiterrs = 0; Nblkerrs = 0; Nblocks = 100;
for blk = 1:Nblocks
    msg = randi([0 1],1,A); %generate random K-bit message
    [quot,rem] = gfdeconv([zeros(1,crcL) fliplr(msg)],crcg);
    msgcrc = [msg fliplr([rem zeros(1,crcL-length(rem))])]; %K bits

    u = zeros(1,N);

    u(Q1(N-K+1:end)) = msgcrc; %assign message bits

    m = 1; %number of bits combined
    for d = n-1:-1:0
        for i = 1:2*m:N
            a = u(i:i+m-1); %first part
            b = u(i+m:i+2*m-1); %second part
            u(i:i+2*m-1) = [mod(a+b,2) b]; %combining
        end
        m = m * 2;
    end
    cword = u;

    s = 1 - 2 * cword; %BPSK bit to symbol conversion
    r = s + sigma * randn(1,N); %AWGN channel I
    %quantization
    r = satx(r,rmax);
```

So once you do that other things do not change too much except, so let us focus in the CRC computation, okay so you have the message that gets generated remember I am generating only A messages now not K and then I do a division you have to divide one polynomial by another so what I am doing here is setting up those polynomials and I am using this function gfdeconv which is available in MATLAB, it is also available in Octave so if through some tool boxes so you can this command will work, so it will divide the first polynomial in binary by this other polynomial.

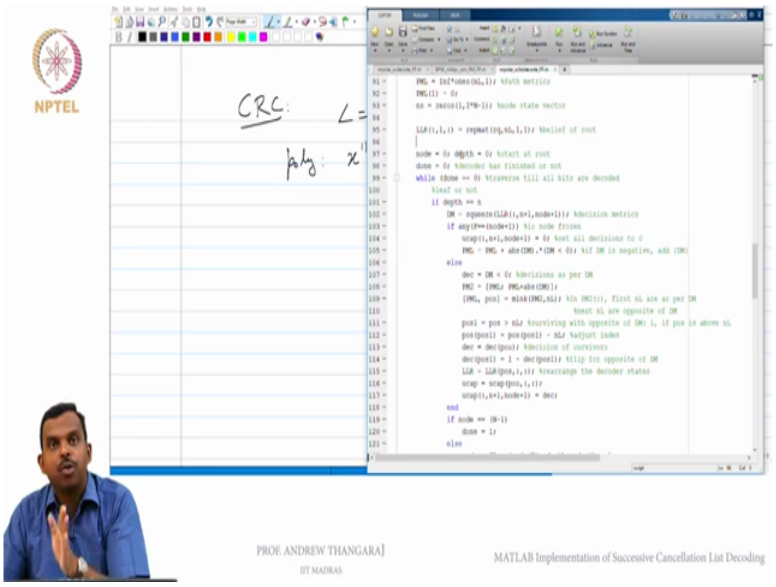
The reason why we do this flipping okay so you saw the flipping even in the CRCG definition if you saw it earlier so even here I flipped, the reason is this gfdeconv function uses values in increasing powers. So you have to do this flipping to make sure that the increasing powers of the polynomial get correctly reflected, the zeros I put here is the remainder okay the remainder will come and occupy the zero positions and this is just the message flipped so that the polynomial becomes the correct degree and then you have that down here.

So you find the remainder, the quotient is actually not very relevant to us the remainder is what matters and the remainder gets added to the CRC part, okay so the message comes as it is and then you will take the remainder and zero and I have done flip LRS as well, okay so that is okay. So that the remainder which is again in increasing powers becomes in decreasing powers, okay so this is how it is done.

So this is described in the standard as well, the only thing to note is I use this gfdeconv to do the division of polynomials, so polynomial division is not very difficult to write in case you do not find it, it is easy to write polynomial division but nevertheless this is we use this function to make a job easy, okay. So you have message and then you have message plus CRC this will be K bits, okay so this will be this you can see I have made it equal to CRCL okay the length I have added enough zeros to make sure that the overall length is CRCL and this will be K bits, okay so maybe I should write that down.

And then after that I just encode CRC so message CRC is what gets assigned as the u value (())(6:28) and then there is no major change as far as the encoding is concerned modulation and quantization, then comes the big change in the SC decoders, okay. So you need NL decoders so whatever how many over less size you want that many decoders you need, okay.

(Refer Slide Time: 6:50)


NPTEL
CRC:
L =
poly: x
PROF. ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation List Decoding

```
PML = Inf*ones(nL,1); %Path metrics
PML(1) = 0;
ns = zeros(1,2*N-1); %node state vector

LLR(:,1,:) = repmat(rq,nL,1,1); %belief of root

node = 0; depth = 0; %start at root
done = 0; %decoder has finished or not
while (done == 0) %traverse till all bits are decoded
    %leaf or not
    if depth == n
        DM = squeeze(LLR(:,n+1,node+1)); %decision metrics
        if any(F==(node+1)) %is node frozen
            ucap(:,n+1,node+1) = 0; %set all decisions to 0
            PML = PML + abs(DM).*(DM < 0); %if DM is negative, add |DM|
        else
            dec = DM < 0; %decisions as per DM
            PM2 = [PML; PML+abs(DM)];
            [PML, pos] = mink(PM2,nL); %In PM2(:), first nL are as per DM
                                       %next nL are opposite of DM
            pos1 = pos > nL; %surviving with opposite of DM: 1, if pos is above nL
            pos(pos1) = pos(pos1) - nL; %adjust index
            dec = dec(pos); %decision of survivors
            dec(pos1) = 1 - dec(pos1); %flip for opposite of DM
            LLR = LLR(pos,:,:); %rearrange the decoder states
            ucap = ucap(pos,:,:);
            ucap(:,n+1,node+1) = dec;
        end
        if node == (N-1)
            done = 1;
        else
```

So these LLR values okay so previously I think I had it as capital L, I have now changed it as LLR such the change in the notation, these LLR values will not just be N plus 1, comma N so you can go and look at the SC decoder this is just the successive cancellation decoder, if you look at the successive cancellation decoder this L was just N plus 1, comma N what I have done in the list decoder?

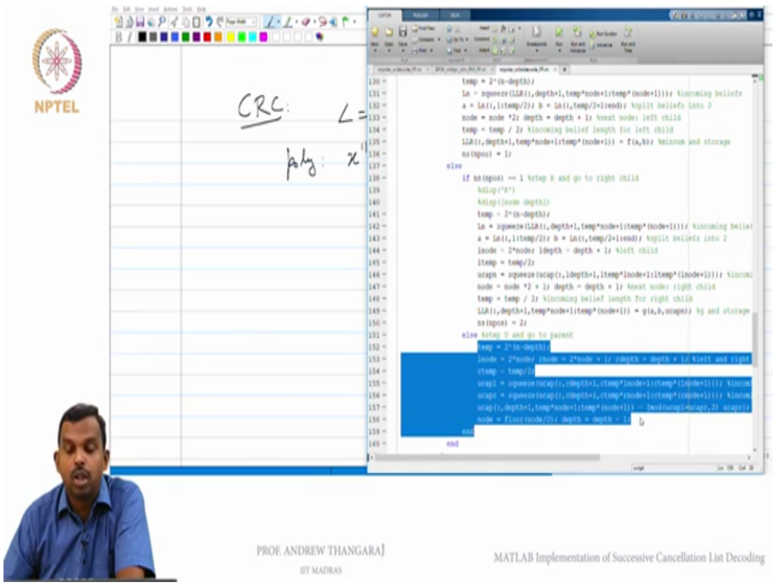
So if you notice this first tab I have is the SC decoder, the last tab that I have is the SC list decoder. In the list decoder these LLR values there is 3 dimensions in the array the first NL gets added, NL, comma N plus 1, comma N for every decoder I need a new array a different array, okay. So we need NL of these u caps also and then I need the path matrix, path matrix I am calling SPML so this path matrix was not there before, we needed now.

If you notice I am initializing the path matrix there are NL of them and all of them I am initializing to infinity, okay and then I am setting the first alone to 0, okay so which means initially only one decoder is active. So if the path matrix is infinity, it means it is not going to be considered by anything, okay so infinity is just the value I use in MATLAB you can use some other value if you do not like it, okay so that is the path matrix for you.

Okay, so this node state vector is actually common to all the decoders, so you do not need to maintain multiple values for node state vector it is the same thing will all the decoders traverse at the same time, they synchronize in that fashion. So initialization at the root, now previously we did only for one decoder so the only one value got initialized now we have to repeat it for NL times. So all the believes of all the decoders are initialized, okay.

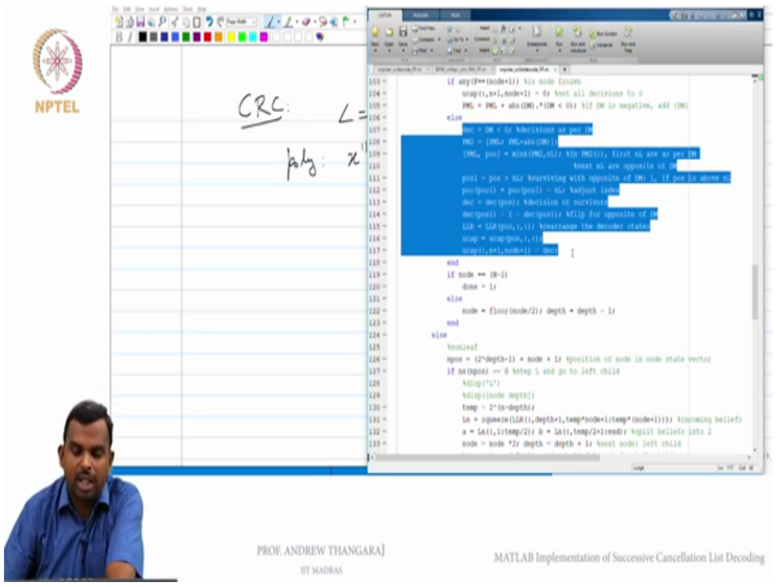
And then after that the decoder actually proceeds in the same way through the node and the depth and moving left down and then going back up and then going right, doing the F operation, doing the G operation, doing the U operation to go up all of them are the same, okay except that you have to repeat it for all the decoders.

(Refer Slide Time: 8:54)



```
                temp = 2^(n-depth);
                Ln = squeeze(LLR(:,depth+1,temp*node+1:temp*(node+1))); %incoming beliefs
                a = Ln(:,1:temp/2); b = Ln(:,temp/2+1:end); %split beliefs into 2
                node = node *2; depth = depth + 1; %next node: left child
                temp = temp / 2; %incoming belief length for left child
                LLR(:,depth+1,temp*node+1:temp*(node+1)) = f(a,b); %minsum and storage
                ns(npos) = 1;
            else
                if ns(npos) == 1 %step R and go to right child
                    %disp('R')
                    %disp([node depth])
                    temp = 2^(n-depth);
                    Ln = squeeze(LLR(:,depth+1,temp*node+1:temp*(node+1))); %incoming belie
                    a = Ln(:,1:temp/2); b = Ln(:,temp/2+1:end); %split beliefs into 2
                    lnode = 2*node; ldepth = depth + 1; %left child
                    ltemp = temp/2;
                    ucapn = squeeze(ucap(:,ldepth+1,ltemp*lnode+1:ltemp*(lnode+1))); %incom
                    node = node *2 + 1; depth = depth + 1; %next node: right child
                    temp = temp / 2; %incoming belief length for right child
                    LLR(:,depth+1,temp*node+1:temp*(node+1)) = g(a,b,ucapn); %g and storage
                    ns(npos) = 2;
                else %step U and go to parent
                    temp = 2^(n-depth);
                    lnode = 2*node; rnode = 2*node + 1; cdepth = depth + 1; %left and right
                    ctemp = temp/2;
                    ucapl = squeeze(ucap(:,cdepth+1,ctemp*lnode+1:ctemp*(lnode+1))); %incom
                    ucapr = squeeze(ucap(:,cdepth+1,ctemp*rnode+1:ctemp*(rnode+1))); %incom
                    ucap(:,depth+1,temp*node+1:temp*(node+1)) = [mod(ucapl+ucapr,2) ucapr];
                    node = floor(node/2); depth = depth - 1;
                end
            end
```

NPTEL
CRC:
L =
poly: x
PROF. ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation List Decoding


NPTEL
CRC:
L =
poly: x
PROF. ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation List Decoding

```
            if any(F==(node+1)) %is node frozen
                ucap(:,n+1,node+1) = 0; %set all decisions to 0
                PML = PML + abs(DM).*(DM < 0); %if DM is negative, add |DM|
            else
                dec = DM < 0; %decisions as per DM
                PM2 = [PML; PML+abs(DM)];
                [PML, pos] = mink(PM2,nL); %In PM2(:), first nL are as per DM
                                           %next nL are opposite of DM
                pos1 = pos > nL; %surviving with opposite of DM: 1, if pos is above nL
                pos(pos1) = pos(pos1) - nL; %adjust index
                dec = dec(pos); %decision of survivors
                dec(pos1) = 1 - dec(pos1); %flip for opposite of DM
                LLR = LLR(pos,:,:); %rearrange the decoder states
                ucap = ucap(pos,:,:);
                ucap(:,n+1,node+1) = dec;
            end
            if node == (N-1)
                done = 1;
            else
                node = floor(node/2); depth = depth - 1;
            end
        else
            %nonleaf
            npos = (2^depth-1) + node + 1; %position of node in node state vector
            if ns(npos) == 0 %step L and go to left child
                %disp('L')
                %disp([node depth])
                temp = 2^(n-depth);
                Ln = squeeze(LLR(:,depth+1,temp*node+1:temp*(node+1))); %incoming beliefs
                a = Ln(:,1:temp/2); b = Ln(:,temp/2+1:end); %split beliefs into 2
                node = node *2; depth = depth + 1; %next node: left child
```

So let me start with the L operation, this is doing the step L and going to the left child. So if you notice nothing much has changed expect that this command is LN was not using the squeeze before but now I am doing the squeeze, so if you look at LLR of colon which means I take the values from all the decoders, not just the one decoder all the NL decoders and then I squeeze out that dimension the depth plus 1 is just the single dimension, so I squeeze it out so I get a 2 dimensional array, this LN will be a 2 dimensional array, okay.

And then I split it into A and B and then everything else is the same and then I assign the F value back into the original array, okay so that is all. So instead of doing it for one decoder, the same operation I repeat for NL decoders at the same time, okay. Same thing with step R also, nothing changes in step R same way we squeeze and we get from the 3 dimensional array we go to a 2 dimensional array to get all the values for a particular depth then you find A, B same thing with u cap N, nothing changes expect that you are doing the same operations for 4 decoders, okay and same thing was step U also, the step U that goes to parent nothing much changes, you do it for all 4 decoders, okay.

The only significant change is in the decision when depth is N when it is a leaf you need to make decisions and there the list decoder has a lot of change, every time you make a decision you have to update a path matrix you will get 8 path matrix you have to choose the best 4 out of the 8 path matrix and throw away and do some rearrangement in the decoders, okay. So that requires some effort, okay in this there is a change there and let me talk about it first.

First is a decision matrix, okay this is the decision matrix for the 4 decoders and first you check if it is frozen, if it is frozen things are easy, if it is frozen there is no need to look at both decisions if you remember from the lecture if the bit is frozen you simply decide (everything is 1) everything is 0 but you update the path matrix this is important, okay path matrix is updated, if the path matrix is negative the absolute value gets added, okay this piece of code does that for you, okay absolute value dot into DM less than 0, so if actually the DM is negative it will get absolute value will get added to the path matrix, okay.

And then after that in case it is not frozen you have to make decisions, okay. I am first deciding according to the decision matrix this DEC is a decision as per the decision matrix, okay. And now what I do is in PM 2 not only do I consider the decision matrix this is I also consider the opposite decision which will make the path matrix increased by absolute value of DM, okay.

So that is what I put here in PM 2 first NL are as per DM, next NL are opposite of decision matrix, if the decision you made in the first NL positions is according to the decision matrix, the next NL in the 8 decoders you are considering the first 4 decoders the decisions you made were as per the decision matrix, the next 4 it is opposite to that, okay. So there are multiple ways to do it, I am doing it in this fashion, okay.

And then I do these adjustments, okay so find out those that survive so this is the important step this is the sorting step okay so this uses min K, what is min K? If you have an array min K will give you the least 4 elements from the array, okay. So remember this PM is all positive values so you do not need to take absolute value anymore, so this min K will give you the least K values from an array.

So this PM 2 has two times NL length okay two times NL if NL is 4, it has length 8 out of which I will take the least 4, okay so you get the new PML which is the least 4 and also the positions of the new PML is in POSS okay which are the least 4? You had 8 values, some of them could be in 1 to 4, some of them could be in 5 to 8 also which came into the least 4 that is captured in POSS, okay.

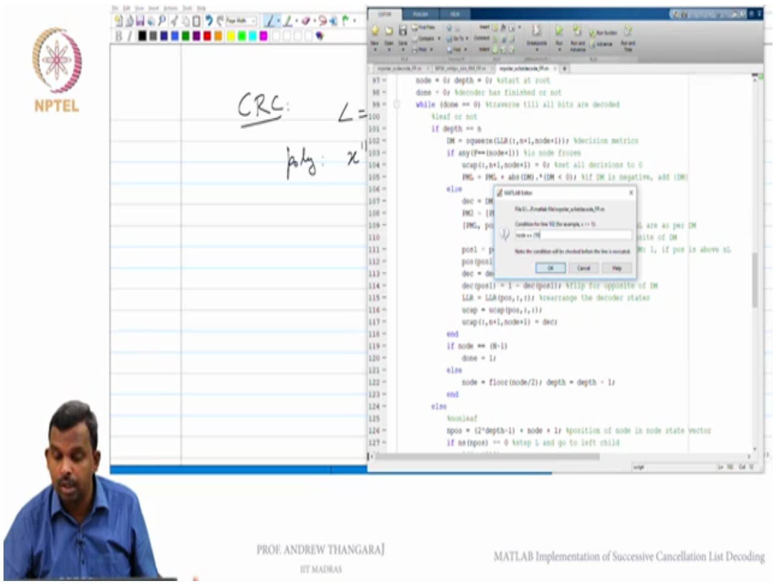
If so I need to know which of the decisions are that survive the decisions that survive are in POSS, okay which survive and they are opposite of DM that I need to know and then I need to subtract this NL to get the actual indices and then permute the decisions as per that and then invert the decisions when the decision is opposite of DM, so these are just routine operations, we will run it once and I will show you how this is working, okay and then I rearrange my decoder states.

So basically I delete the decoders that are not valid and keep the decoders that are valid and then assign this to decision, okay so this little piece of code actually implements the list stage okay so expanding the list from 4 to 8 and then contracting it again back to 4, okay. So you go to 8, you consider both decisions when the bit is not frozen, arrange the path matrix, pick the best 4 of the path matrix and then do the adjustments.

So the adjustments are little bit complicated I will let you look at the code and think about that the code is there with you if you run it once or twice you will see how the code works and you will understand what I mean, okay so it is a little bit involved I do not want to go into great detail on the code, okay but this part implements exactly that, okay. So if you have version of MATLAB or Octave I will let you run it for a while with some small length maybe and then look at the look at what this piece of code is doing and then you will learn little bit more about how this is doing and clearly this is not the only way to do it, there are multiple ways to implement this if you have a better way you are welcome to implement this as well, okay.

So this is the implementation of the list decoder, please look at this part of the code and try to see it if this is correct if I made any mistakes or if you think you can improve this in a nice way, okay. So that is the implementation of the list decoder, okay.

(Refer Slide Time: 14:50)

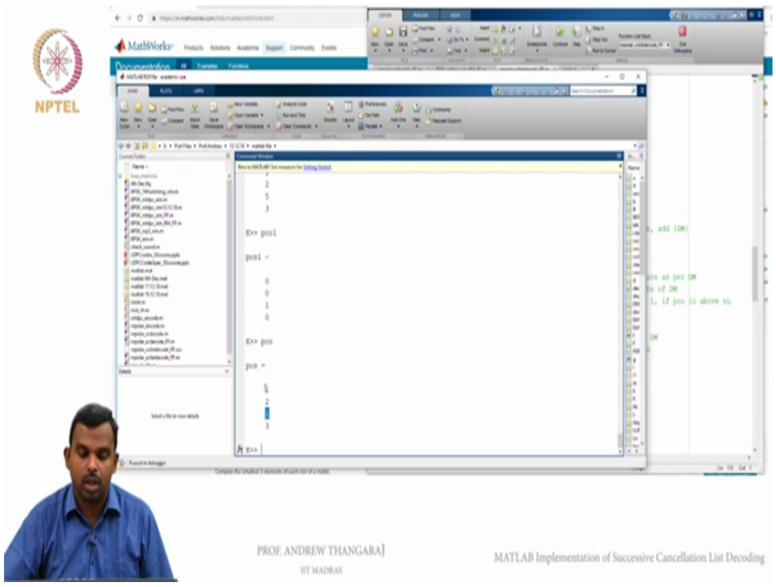


```
node = 0; depth = 0; %start at root
done = 0; %decoder has finished or not
while (done == 0) %traverse till all bits are decoded
    %leaf or not
    if depth == n
        DM = squeeze(LLR(:,n+1,node+1)); %decision metrics
        if any(F==(node+1)) %is node frozen
            ucap(:,n+1,node+1) = 0; %set all decisions to 0
            PML = PML + abs(DM).*(DM < 0); %if DM is negative, add |DM|
        else
            dec = DM [illegible]
            PM2 = [P[illegible]
            [PML, po[illegible]                              [illegible]L are as per DM
                                                             [illegible]site of DM
            pos1 = p[illegible]                              [illegible]M: 1, if pos is above nL
            pos(pos1[illegible]
            dec = de[illegible]
            dec(pos1) = 1 - dec(pos1); %flip for opposite of DM
            LLR = LLR(pos,:,:); %rearrange the decoder states
            ucap = ucap(pos,:,:);
            ucap(:,n+1,node+1) = dec;
        end
        if node == (N-1)
            done = 1;
        else
            node = floor(node/2); depth = depth - 1;
        end
    else
        %nonleaf
        npos = (2^depth-1) + node + 1; %position of node in node state vector
        if ns(npos) == 0 %step L and go to left child
```

File I:...\Matlab files\sopolar_sclistdecode_FP.m
Condition for line 102 (for example, x == 5):
node == [illegible]
Note: the condition will be checked before the line is executed.

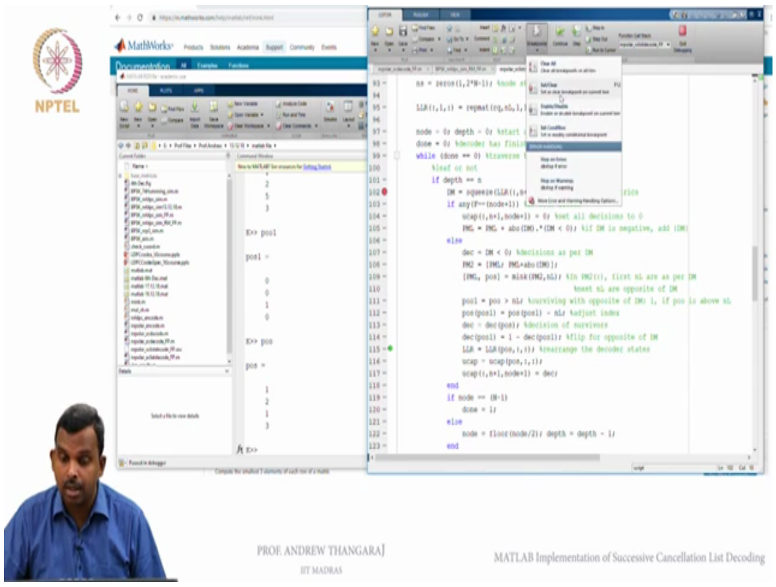
PROF. ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation List Decoding



```
    2
    5
    3

K>> pos1

pos1 =

    0
    0
    1
    0

K>> pos

pos =

    1
    2
    1
    3

K>>
```

PROF. ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation List Decoding



```
ns = zeros(1,2*N-1); %node st

LLR(:,1,:) = repmat(rq,nL,1,1

node = 0; depth = 0; %start
done = 0; %decoder has finis
while (done == 0) %traverse
    %leaf or not
    if depth == n
        DM = squeeze(LLR(:,n+                    rics
        if any(F==(node+1))
            ucap(:,n+1,node+1) = 0; %set all decisions to 0
            PML = PML + abs(DM).*(DM < 0); %if DM is negative, add |DM|
        else
            dec = DM < 0; %decisions as per DM
            PM2 = [PML; PML+abs(DM)];
            [PML, pos] = mink(PM2,nL); %In PM2(:), first nL are as per DM
                                        %next nL are opposite of DM
            pos1 = pos > nL; %surviving with opposite of DM: 1, if pos is above nL
            pos(pos1) = pos(pos1) - nL; %adjust index
            dec = dec(pos); %decision of survivors
            dec(pos1) = 1 - dec(pos1); %flip for opposite of DM
            LLR = LLR(pos,:,:); %rearrange the decoder states
            ucap = ucap(pos,:,:);
            ucap(:,n+1,node+1) = dec;
        end
        if node == (N-1)
            done = 1;
        else
            node = floor(node/2); depth = depth - 1;
        end
```

So now what I want to do is maybe run this once I will run it once, now when you run it it is good to stop at a particular place, now MATLAB's (decoder) debugger allows all these kind of nice and simple conditions to make you stop somewhere, so I will stop at some point let us say 250, so at the 250th node I want to stop, so I will set a conditional breakpoint there and then I will run it, okay (how many blocks I am running) I think it is enough if I run one block, okay, save I will run, okay.

Brief warning about min K, so I believe MATLAB introduce this in 2018 onwards in case you have a older version you have to write your own min K, so min K is not very hard to write here is a simple version min K of x comma K is simply sorted and take the first K so this is actually not very efficiently that is the algorithm which will find the minimum K without sorting the whole list, but anyway so for quick and dirty implementation this is good, okay.

I have set my breakpoint here and now I am going to run this, so we can implement our own version of min K you need to return the minimum K elements and where they are also, okay. So that requires a little bit of coding and you can do that with sort, sorts it in ascending order you pick the first K and return it and pick the first K positions and return it okay so this will should work and this is your own min K and you can use MATLAB's min K as well, it is probably definitely more efficient if you have 2018B or later I think that will work, okay.

So let us run this, so it is worked and it is gone upto node of node will be 250 like I said so let us look at so what happens at this point, so interesting to look at the path matrix I am sorry PML I think was the path matrix 118, 149, 149, 149 this is the path matrix at this point, okay and you find the decision matrix and see if it is frozen it is not frozen, okay so you went into the decision, okay.

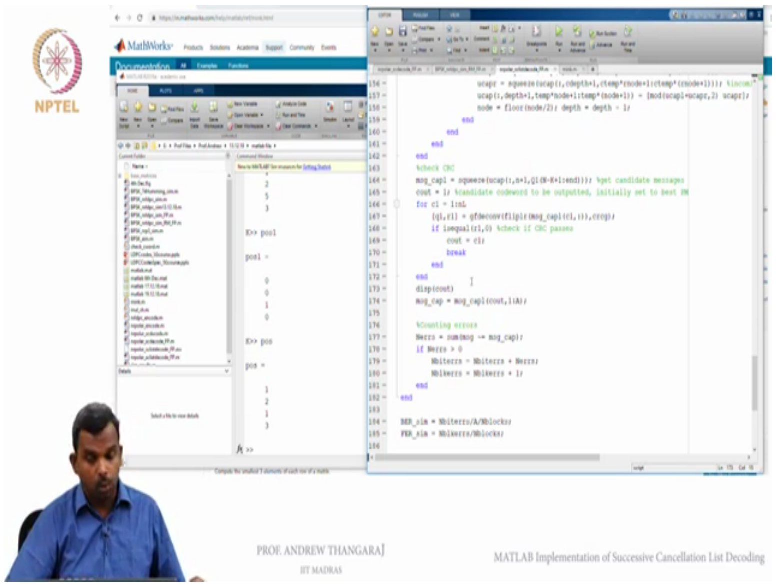
So let us look at what the decision matrix was, 31 minus 6310 so that is the decision matrix at this point, okay you look at the decisions as per the decision matrix you will see this will be 0 1 0 0, okay and then you look at the PM2, okay so notice what PM 2 is, okay it is the path matrix if you decide as per the decision matrix and the path matrix if you decide opposite to the decision matrix, okay so you add the DM absolute value, okay and then you sort it, you have min K.

So if you do the min K you get the new PML which is you see that 149 is the one that is one out it is nothing no major churn that has happened here and POSS is just 1 2 3 4, so everything is easy, so maybe what when you can see how this is done, so this is not very instructive so maybe I will setup a breakpoint here and then continue to get to the next one so you see PM PML is the same thing but hopefully the decision matrix is little bit more interesting let us see if the POSS became a bit more interesting or it is again 1 2 3 4, so if it is just 1 2 3 4 then you are not doing a lot of sorting and moving around is not very nice.

So maybe I will continue for a few more things and then step in and see okay I think maybe we are in luck okay. So let us look at okay. So here is an interesting situation where POSS became 1 2 5 3 so your PML PM 2 look at so this was the PM 2 188 219 223 all these things. So when you sorted you got 188 first and then 219, another 219 and then 223, so if you look at the positions 5 got added so we got 1 2 3 but 5 got added. So you made a decision opposite to the decision matrix.

So when you do that you have to find the places where this POSS 1 happened, so POSS 1 will indicate to you the positions where decisions opposite to the DM was done this was in the third position so you get 0 0 1 0 and those alone you have to subtract NL so that you get back POSSES remaining one. So what happens here is the first decoder had both decisions surviving, first decoders decision as per its decision matrix and against its decision matrix also survived, okay and then 2 and 3 had decisions as per their decision matrix surviving, okay and then you did that and then see remember decision was according to the decision matrix so now I have to permute it as well because some of the fourth decoder for instance is not even their so that needs to go and then I have to flip the POSS 1, after that you just continue as it is, okay. So that was one example to show you how this decision happens.

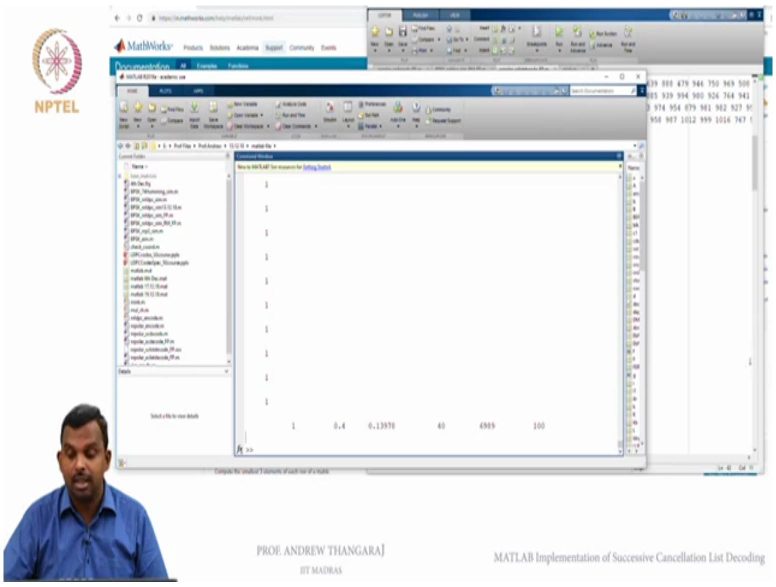
(Refer Slide Time: 20:36)



```
     2
     5
     3

K>> pos1

pos1 =

     0
     0
     1
     0

K>> pos

pos =

     1
     2
     1
     3

fx >>
```

```
                    ucapr = squeeze(ucap(:,cdepth+1,ctemp*cnode+1:ctemp*(cnode+1))); %incom
                    ucap(:,depth+1,temp*node+1:temp*(node+1)) = [mod(ucapl+ucapr,2) ucapr];
                    node = floor(node/2); depth = depth - 1;
                end
            end
        end
    end
    %check CRC
    msg_capl = squeeze(ucap(:,n+1,Q1(N-K+1:end))); %get candidate messages
    cout = 1; %candidate codeword to be outputted, initially set to best PM
    for c1 = 1:nL
        [q1,r1] = gfdeconv(fliplr(msg_capl(c1,:)),crcg);
        if isequal(r1,0) %check if CRC passes
            cout = c1;
            break
        end
    end
    disp(cout)
    msg_cap = msg_capl(cout,1:A);

    %Counting errors
    Nerrs = sum(msg ~= msg_cap);
    if Nerrs > 0
        Nbiterrs = Nbiterrs + Nerrs;
        Nblkerrs = Nblkerrs + 1;
    end
end

BER_sim = Nbiterrs/A/Nblocks;
FER_sim = Nblkerrs/Nblocks;
```

NPTEL
PROF. ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation List Decoding



```
     1

     1

     1

     1

     1

     1

     1

     1

     1

     1

         1         0.4       0.13978         40        6989        100

>>
```

NPTEL
PROF. ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation List Decoding

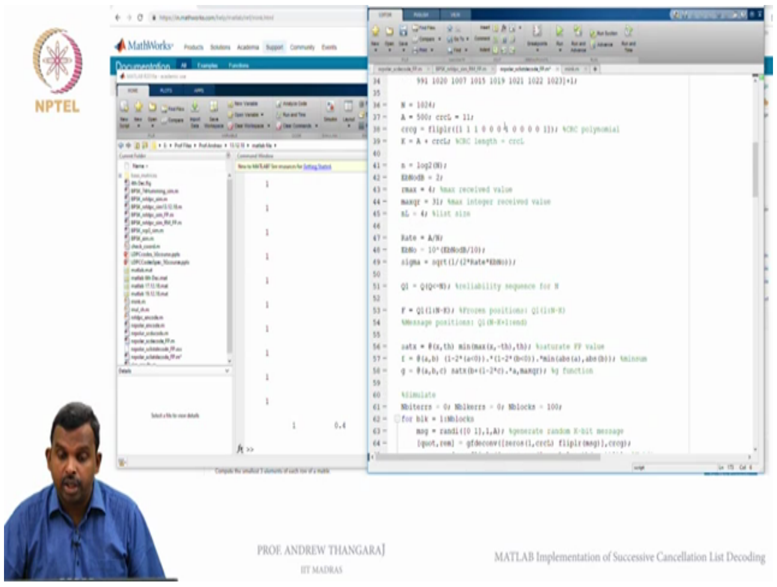
And then what do you do after everything is done you have to check your CRC so let me just run to cursor here I have come here all my decoding is complete I am looking at the candidates. So I am taking at taking all the candidates you will have NL candidates all of them will be in message capl, okay and then you have to see which candidate is to be outputted you go through all the candidates and then you do the division again to check which CRC is satisfied or not.

So let us go through this, so I am going and checking if the CRC is satisfied or not, if R1 is 0 CRC is satisfied and it passes yes it passes so you are done, okay. So it might happen that this R1 is not 0 also but in most cases the CRC will pass and you get through and you get the decoder, okay. So hopefully that showed you how the decoder works and maybe you can for instance if you are curious to see what the cout is you can print cout I am sorry print does not work so you can display cout and then run the decoder for a little while and you will see whether or not other decisions are getting done here so let me adjust the eb over n not to something like say 1.5 and then do 100 blocks okay and remember I am outputting the cout which candidate code word got chosen, okay.

So if you run it so you see 1 comes quite often most of the time there was the 2, the second candidate got chosen there once in a while we do not expect we always expect the least path matrix candidate to win, but once in a while you will get something lower also, okay so that can happen maybe we will go to a lower SNR maybe you will see more of it, okay. For most of the time the best path matrix wins, once in a while you get the second path matrix also winning, okay very rarely, okay here we got one more, here we got one more, here we got three, okay so once in a while when the decoder happens the second or third candidate code word passes the CRC and other thing does not pass, okay.

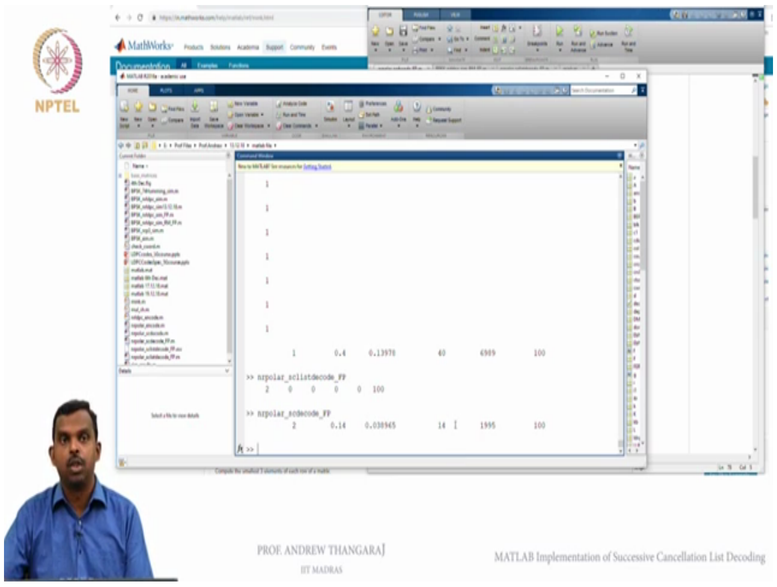
So if you want to do repeat the simulation that we have remember this is roughly rate half and we saw 2 dB was the number where the LDPC code was giving you 0 errors for 100 code words, let me stop the display of the cout this is not needed anymore, okay.

(Refer Slide Time: 23:17)



```
        991 1020 1007 1015 1019 1021 1022 1023]+1;

N = 1024;
A = 500; crcL = 11;
crcg = fliplr([1 1 1 0 0 0 1 0 0 0 0 1]); %CRC polynomial
K = A + crcL; %CRC length = crcL

n = log2(N);
EbNodB = 2;
rmax = 4; %max received value
maxqr = 31; %max integer received value
nL = 4; %list size

Rate = A/N;
EbNo = 10^(EbNodB/10);
sigma = sqrt(1/(2*Rate*EbNo));

Q1 = Q(Q<=N); %reliability sequence for N

F = Q1(1:N-K); %Frozen positions: Q1(1:N-K)
%Message positions: Q1(N-K+1:end)

satx = @(x,th) min(max(x,-th),th); %saturate FP value
f = @(a,b) (1-2*(a<0)).*(1-2*(b<0)).*min(abs(a),abs(b)); %minsum
g = @(a,b,c) satx(b+(1-2*c).*a,maxqr); %g function

%Simulate
Nbiterrs = 0; Nblkerrs = 0; Nblocks = 100;
for blk = 1:Nblocks
    msg = randi([0 1],1,A); %generate random K-bit message
    [quot,rem] = gfdeconv([zeros(1,crcL) fliplr(msg)],crcg);
```

PROF. ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation List Decoding



```
     1

     1

     1

     1

     1

           1          0.4      0.13978          40         4989         100

>> nrpolar_sclistdecode_FP
     2     0     0     0     0   100

>> nrpolar_scdecode_FP
           2         0.14     0.038965          14         1995         100

>>
```

PROF. ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation List Decoding

So let us just see if 2 dB for 100 code words for rate half for the polar code list decoder with 4 how does that perform, does it give you 0 or not? It will give you something better than the SC decoder at 2 dB that gave about 12 errors out of 100 if you remember correctly, this should give you something better let us see, so it gives you 0, okay so you see in these 100 code words the list decoder, decoder N gave you 0, if you actually run for the SC decoder at 2 dB which we did a little while ago 2 dB with 100 blocks you will see this will give you some errors, it will not give you 0, it give you 14 errors out of 100. So you see there is an improvement in the list decoder, okay.

So we see that the list decoder really improves in performance by that little bit to make it comparable maybe with the other decoders. So in an ensuing lecture the final lecture possibly of this course we will compare this rate half-length 1000 code between LDPC and polar and then run simulations and compute I mean show you the simulation results and I will show you how you can compare these three codes and decoders and make a nice plot of ber versus eb over n not and make a fair comparison, okay with that we will conclude this course, okay thank you very much.