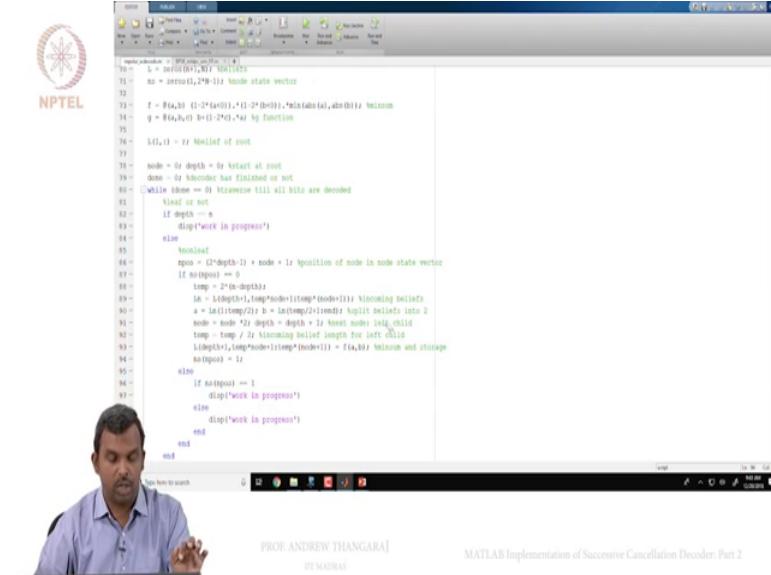


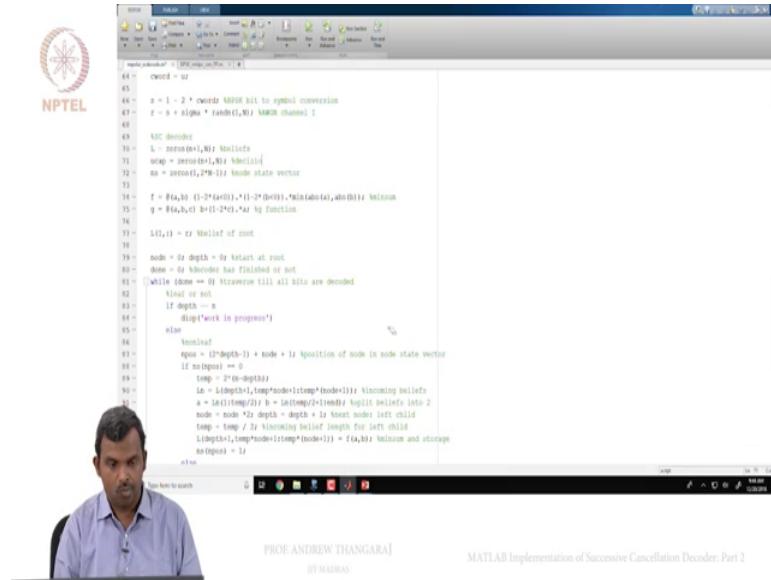
LDPC & Polar codes in 5G Standard
Prof. Andrew Thangaraj
Department of Electrical Engineering
Indian Institute of Technology Madras
MATLAB Implementation of Successive Cancellation Decoder Part 2

(Refer Slide Time: 0:16)



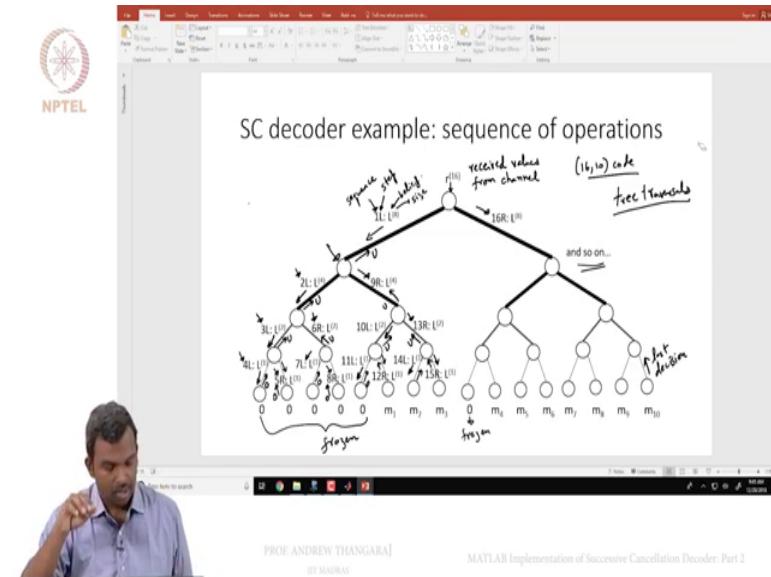
Okay let us continue our coding and you see what the coding involves right, so you have the storage and you have to pull it from storage, pull what you need from storage to the processing and push it back in to the storage okay, so that is how the storage operates, I need to at just this lengths and all that is 2^N minus depth and all that interest a picture, that is not very difficult to do as we can see, of course there are many more efficient ways in which this can be done, I am not focusing on that okay.

(Refer Slide Time: 0:45)



So next is then the state is at, when the node state is at 1 which means the left is already done okay and it is receiving the decision okay, so if you remember this U will flow back also okay and one needs to remember that also okay so because that is important, I did not have the storage for the U cap and that is also the same as this okay, so beliefs and then I have the U cap also okay.

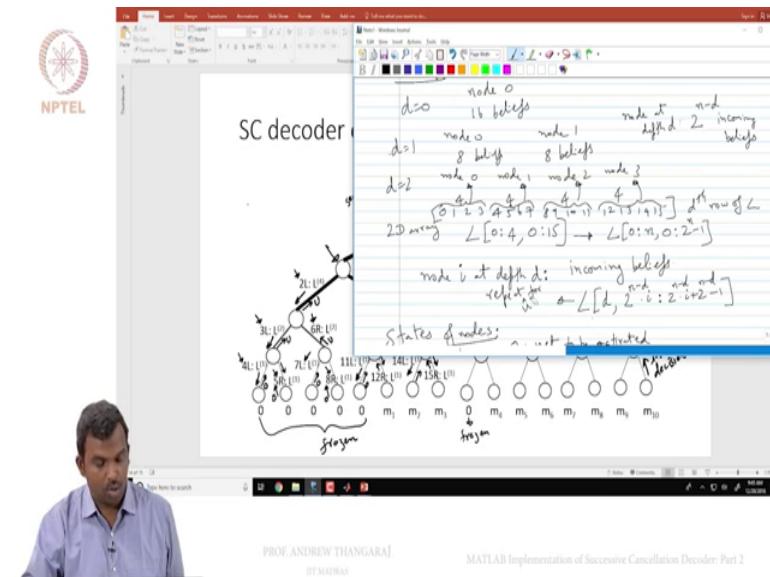
(Refer Slide Time: 1:25)



So once again let me emphasise where this is coming from okay, so this is the decoder, you remember this decoder right, so we stored the incoming beliefs, now what about the decisions that are flowing back? Those also need to be stored okay and you have to restore them and then process them later also, so you have to keep them, so for the decisions also will use the

same sort of a array structure as we used for L okay, we had L is a big array which stores all the beliefs, likewise will have U cap as the array which stores all the decisions that are flowing back okay and the position is exactly the same as L, wherever you computed the Ls position the same place you will put the U position also okay, so that is the idea.

(Refer Slide Time: 2:12)



So maybe I go back to this thing and show you how I organise the L, you remember how we organise L right, so it is the same way will organise the U cap also, the same way in which, so the same way U cap also okay, repeat for U cap okay, so that is what I am going to do, so let me not spend too much time on it we can start coding.

(Refer Slide Time: 2:35)

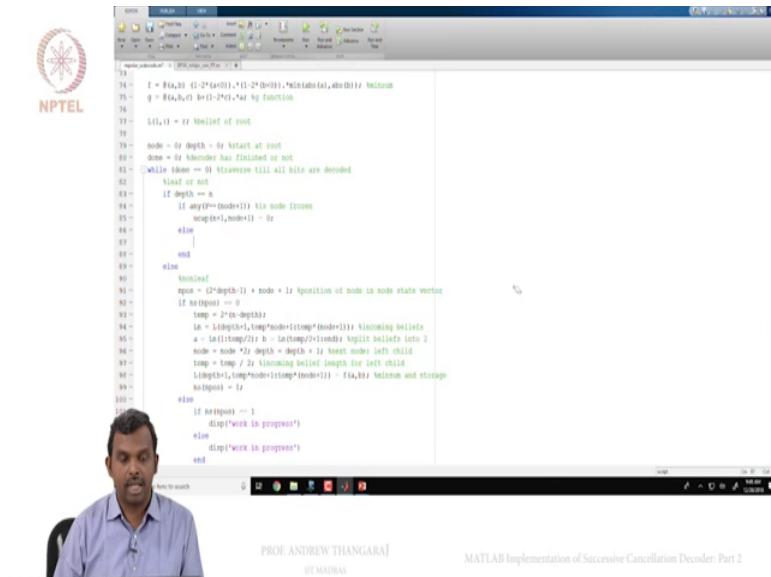


```
function [node] = SCD(node, P, R)
% SCD.m: Successive Cancellation Decoder
% Inputs: node - Node index, P - Power vector, R - Received signal
% Output: node - Decoded node index
%
% This function performs successive cancellation decoding on a
% binary tree. It starts at the root node and decodes each bit sequentially
% until all bits are decoded. The function uses a while loop to iterate
% through each bit position. Inside the loop, it checks if the current
% node is a leaf node. If it is, it returns the node index. If not, it
% performs a search operation to find the next child node. It then
% updates the power vector and received signal for the next iteration.
%
% The function also includes a progress indicator 'work in progress'
% which is displayed in the command window during decoding.

% Initialize variables
P = P'; % Ensure P is a column vector
R = R'; % Ensure R is a column vector
node = 1; % Start at root node
depth = 0; % Initialize depth counter
done = 0; % Initialize done flag
bitmask = 1; % Initialize bitmask for current bit position

% Main loop
while done == 0 % Traverse till all bits are decoded
    if ~isleaf(node) % If not leaf
        if depth == 0 % If first bit
            disp('Work in progress');
        else
            disp('Work in progress');
        end
    end
    if node == 1 % If root node
        node = search(P, R);
    else
        node = search(P, R);
    end
    P = P * node; % Update power vector
    R = R * node; % Update received signal
    depth = depth + 1; % Increase depth
    if depth == 3 % If 3rd bit
        done = 1; % Set done flag
    end
end
% Return decoded node index
node = node;

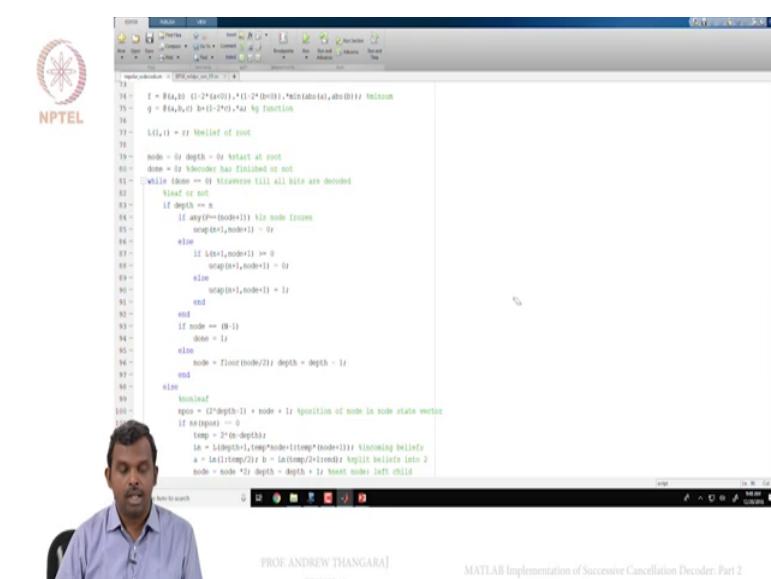
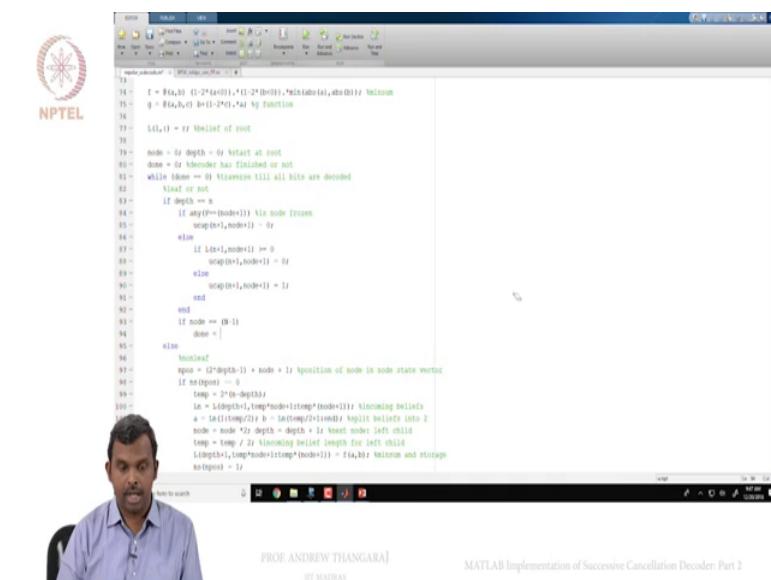
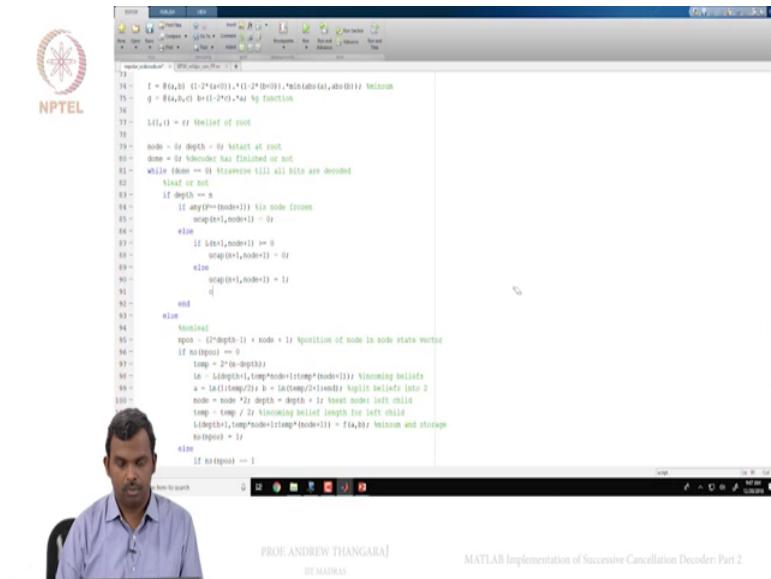
```



So U cap is like that okay, so U cap comes into operation when you either make a decision at the leaf right of you do a step U or even a step R right, even in step R the incoming U cap from the child node is used okay, so maybe I will start with what happens at depth N and how U cap is used here okay, so at depth N I am going to check if node plus 1 is frozen or not okay, so how do I check if node plus 1 is frozen, there is a command you can use here any of F equals equals node plus 1 okay.

So this is node frozen, remember I am at depth N and I am at bottommost thing okay, so I am at the bottom most, the bottommost is easy to address the position in the U cap array, if it is frozen, so I will just explicitly store U cap to be 1 node plus 1 equals 0 okay, so in the bottommost you have the entire length coming in okay and then every node is just one position away, you do not have to do any calculation for the position, that is one thing nice.

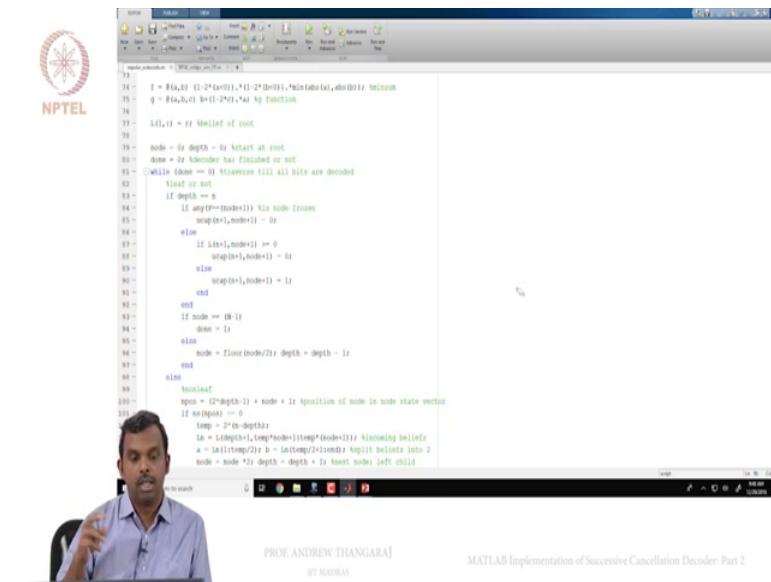
(Refer Slide Time: 4:03)



So here if, so this is the decision right, so if L of N plus 1, node plus 1 is greater than 0, greater and equal to 0, U cap of N plus 1, node plus 1 equals to 0 else U cap of N plus 1, node plus 1 equal to 1 that is, so that is the decision okay, so once you finish the decision, you are not done, you have to handover to the parent right, so how do you hand over to the parent, I do node equals.

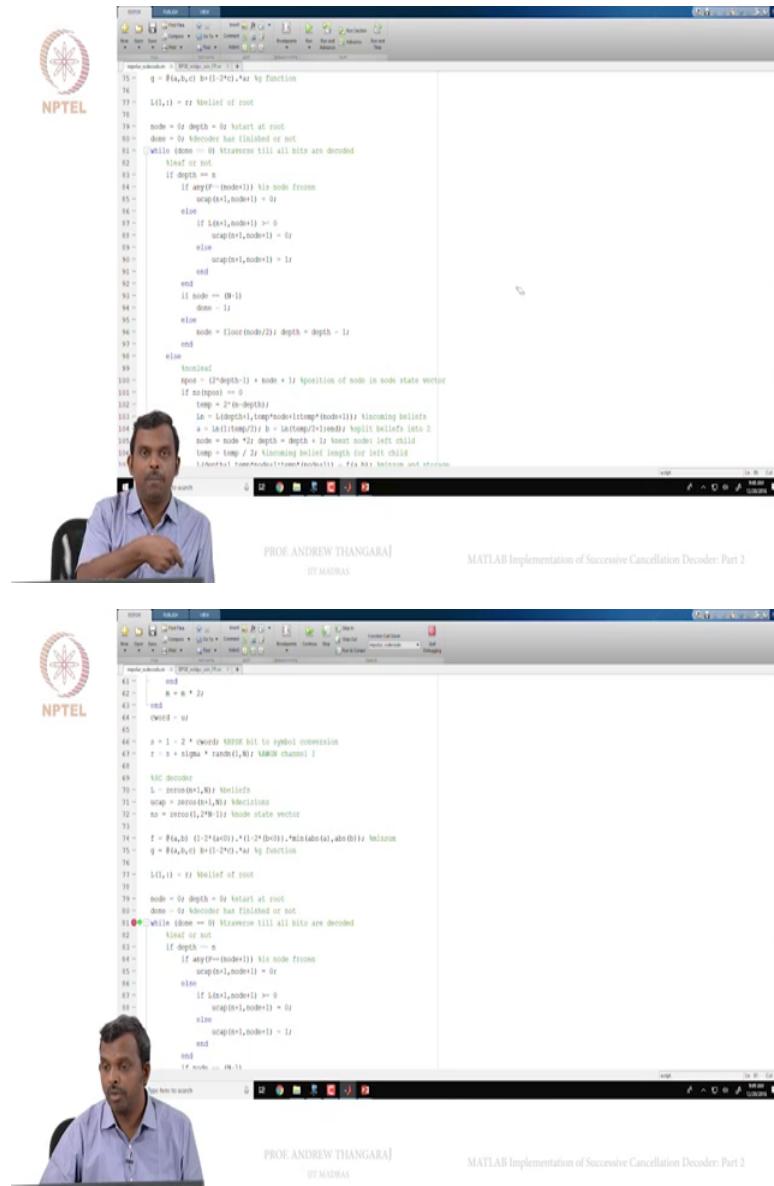
So one thing you can do is you can check if this is this is the last one, if the node equal to equal to N minus 1, so if you have made a decision on the last node, you can set done equals 1 else if you are not done the last one you can make node as floor of node by 2 okay, so this is my flooring and then depth is depth minus 1, so you proceed further okay, so this is going of to the next one.

(Refer Slide Time: 5:24)



I am not going to bother too much about the changing the state of the leaf node okay because it is not so important, once you get there you design and you go on okay, so this is what the leaf node does? Once you come to the leaf node it will go through okay.

(Refer Slide Time: 5:40)



So in fact we have done enough so that even simulate and see whether things are happening correctly or wrongly, it is not too bad at this point I believe you can go up to some point because remember you do a lot of down steps initially and then you make a decision okay and then we can avoid the work in progress quite a bit if you do this, so let us, I think it is good at this point to run it and then see what happens, so let me come up with this point and set up a breakpoint here hopefully it will run and let us run okay when you run it at debugger mode we have to go back to the okay.

(Refer Slide Time: 6:19)



NPTEL

```
70 - l = zeros(1,M); b1 = zeros(1,M);
71 - wcap = zeros(1,M); b2 = zeros(1,M);
72 - n = zeros(1,M-1); b3 = zeros(1,M-1);
73 - g = f(x,b,c) b-(l-2*c)*x' kg Function
74 -
75 - l(l,1) = rx % belief of root
76 -
77 - node = 0 % depth = 0 at start at root
78 - done = 0 % decoder has finished or not
79 - while done == 0 % traverse till all bits are decoded
80 -     if depth == n
81 -         if any(~(node+l)) % if node frozen
82 -             wcap(l:(node+l)) = 0;
83 -         else
84 -             if l(n), (node+l) >= 0
85 -                 wcap(n+1:(node+l)) = 1;
86 -             else
87 -                 wcap(n+1:(node+l)) = 0;
88 -             end
89 -         end
90 -     end
91 -     if node == (N-1)
92 -         done = 1;
93 -     else
94 -         node = floor(node/2); depth = depth + 1;
95 -     end
96 - end
97 - if nargout == 1
98 -     spon = (2^(depth-1) * node + 1) A position of node in node state vector
99 - end
```

PROF. ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 2



NPTEL

```
72 - if depth == n
73 -     if any(~(node+l)) % if node frozen
74 -         wcap(l:(node+l)) = 0;
75 -     else
76 -         if l(n), (node+l) >= 0
77 -             wcap(n+1:(node+l)) = 0;
78 -         else
79 -             wcap(n+1:(node+l)) = 1;
80 -         end
81 -     end
82 - end
83 - if node == (N-1)
84 -     done = 1;
85 - else
86 -     node = floor(node/2); depth = depth + 1;
87 - end
88 - if nargout == 1
89 -     spon = (2^(depth-1) * node + 1) A position of node in node state vector
90 - end
91 - if ~done
92 -     if nargout == 0
93 -         temp = 2^(n-depth);
94 -         lb = Ldepth*(1,temp)*node+ltemp*(node+l)); binominal beliefs
95 -         a = Ldepth*(1,temp)*node+ltemp*(node+l)); binomial beliefs into 2
96 -         node = node*2^depth + a; % next node's next child
97 -         temp = temp / 2; binomial belief length for last child
98 -         Ldepth*(1,temp)*node+ltemp*(node+l)) = ffa,bff binomial and other
99 -         nargout = 1;
100 -     else
101 -         if nargout == 1
102 -             disp('work in progress')
103 -         else
104 -             disp('work in progress')
105 -         end
106 -     end
107 - end
```

PROF. ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 2



NPTEL

```
72 - if depth == n
73 -     if any(~(node+l)) % if node frozen
74 -         wcap(l:(node+l)) = 0;
75 -     else
76 -         if l(n), (node+l) >= 0
77 -             wcap(n+1:(node+l)) = 0;
78 -         else
79 -             wcap(n+1:(node+l)) = 1;
80 -         end
81 -     end
82 - end
83 - if node == (N-1)
84 -     done = 1;
85 - else
86 -     node = floor(node/2); depth = depth + 1;
87 - end
88 - if nargout == 1
89 -     spon = (2^(depth-1) * node + 1) A position of node in node state vector
90 - end
91 - if ~done
92 -     if nargout == 0
93 -         temp = 2^(n-depth);
94 -         lb = Ldepth*(1,temp)*node+ltemp*(node+l)); binominal beliefs
95 -         a = Ldepth*(1,temp)*node+ltemp*(node+l)); binomial beliefs into 2
96 -         node = node*2^depth + a; % next node's next child
97 -         temp = temp / 2; binomial belief length for last child
98 -         Ldepth*(1,temp)*node+ltemp*(node+l)) = ffa,bff binomial and other
99 -         nargout = 1;
100 -     else
101 -         if nargout == 1
102 -             disp('work in progress')
103 -         else
104 -             disp('work in progress')
105 -         end
106 -     end
107 - end
```

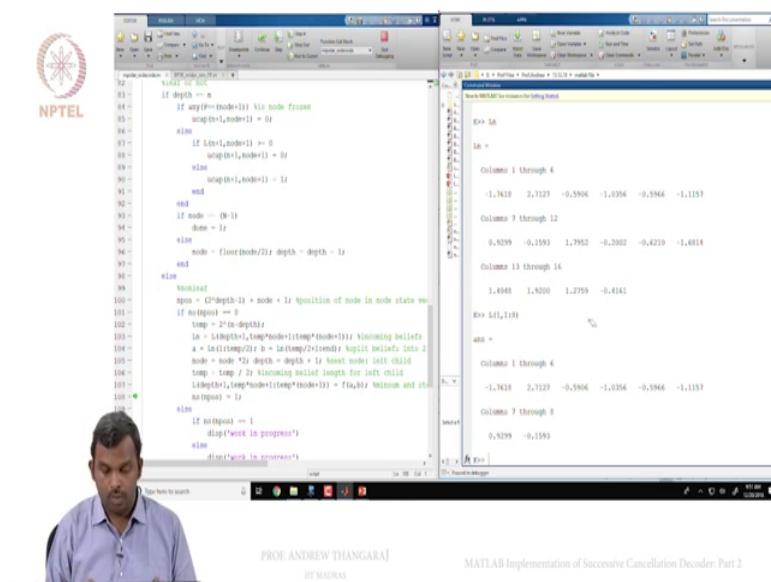
PROF. ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 2

So we have done and this is started, remember our starting point was this is N equals 16 right, yes 16 and 10, so maybe we can see the frozen positions just to be sure, 1, 2, 3, 5, 9, 4 it is okay, so that is the frozen positions, so let us that going in okay, so we step here and the node and depth they should not happen is not a leaf okay and then it is calculate the position, depth in 0, node is 1 so N Poss should work out as 1, so this will be to okay anyway so I check it out N Poss is 1 that is okay and NS of N Poss will be 0, so no problem here and you compute temp, temp will be 16 okay and then depth the 0, so node 0, temp is 16.

So this will be 1 all the way up to 16 okay, so LN will be the same as L, so if we see LN it will have 16 values and this is the received values from the channel okay and then you store temp by 2, A and B, A will be the first part, B will be the second part and then you go to the next node, node is again 0 but depth is 1 okay and temp is become temp by 2 and then you do the storing of L okay.

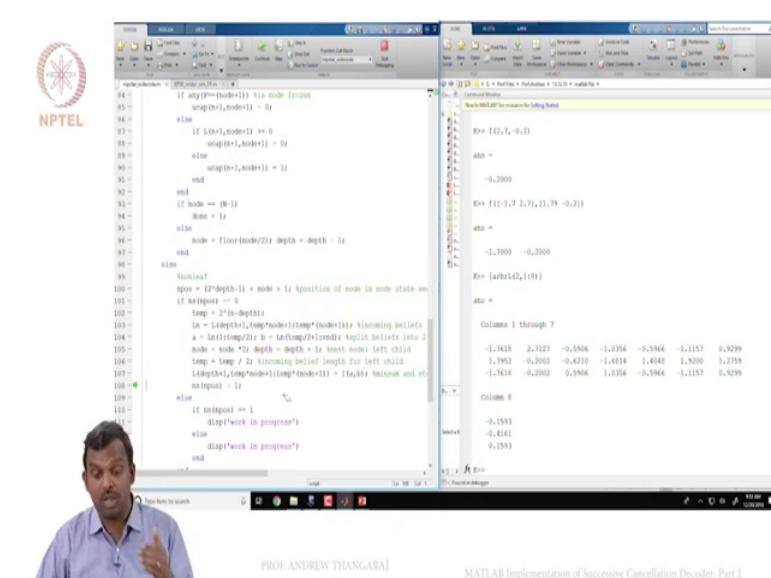
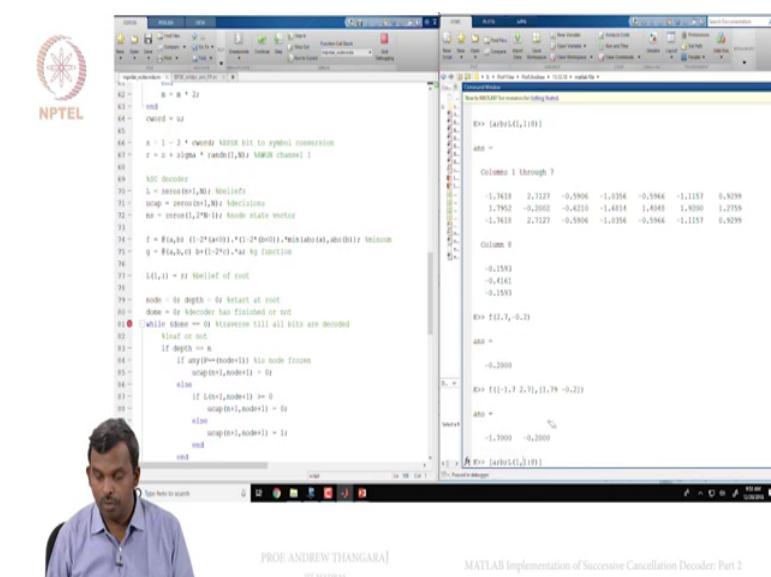
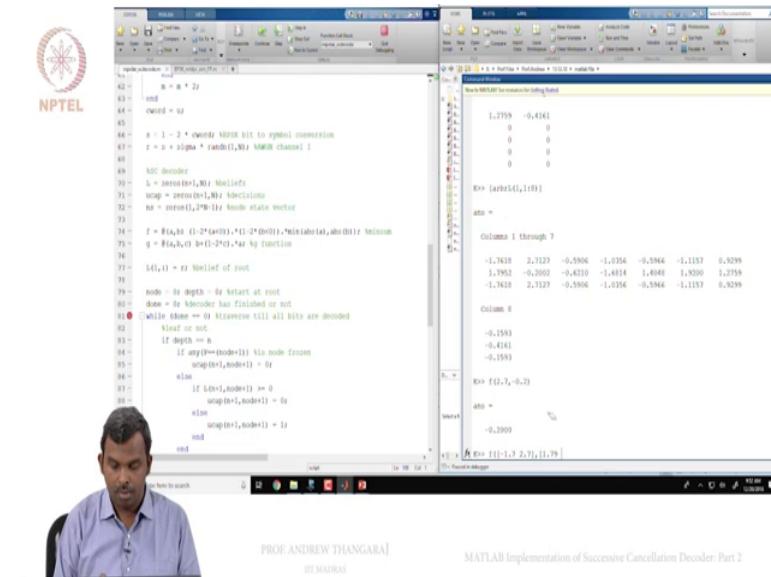
(Refer Slide Time: 7:49)



The image shows a dual-monitor setup. The left monitor displays a video of Prof. Andrew Thangara, a man with dark hair and a beard, wearing a light blue shirt, sitting at a desk and speaking. The right monitor displays the MATLAB graphical user interface. In the MATLAB window, there is a code editor containing MATLAB code for a Successive Cancellation Decoder, a command window showing the output of the code, and a figure window showing a matrix of numerical values.

So let us see L of 1, 1 colon 8 right so this is what I will know, the remaining will actually be 0 and L, so we will just do this and how do you see this, so maybe we can see a little bit better, we do this, so we can see A, we can see B and then we can see L okay, maybe we can see all okay, so this became a bit ugly, so let us see A, B and then L of 1, 1 colon 8 okay, so you can see the minsum is working correctly, so you the minsum is working not so correctly, why is that? Okay so I think, so something is wrong with this minsum, I do know why A and B together should give you this because this minsum is supposed to be minus 0.2 is not it, so something slightly wrong in my F construction.

(Refer Slide Time: 8:59)



So let me just check this out, so if I do F of 2.7 and minus 0.2, I should get minus 0.2 but that seems to be working okay but F of let me check this minus 1.7, 2.7, 1.79 minus 0.2 let see, this is also working out okay oh okay, so I think I know what mistake I made an error, this is L of 2, okay sorry okay, so I was right program was working correctly, I was just not checking the correct L , it needs to go to depth plus 1 right, so the next storage.

(Refer Slide Time: 9:56)

The image shows a dual-monitor setup. The left monitor displays a MATLAB window with a code editor containing C++ code for a successive cancellation decoder. The right monitor displays another MATLAB window showing a command-line interface with various parameters and results. A video feed of a man speaking is overlaid at the bottom left of the screen.

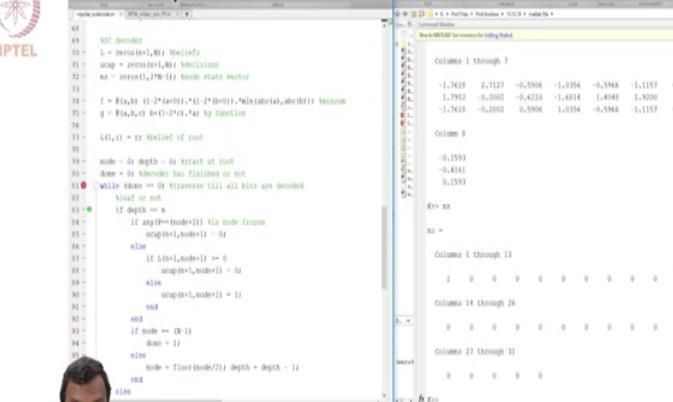
PROF. ANDREW THANGARAJ

IT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 2

So incoming in the next node and that is coming out correctly okay, it is a minsum, you can see the product of the signs and the lower of the two values being retained here okay, so this is the belief here and the state goes into the next one okay, so there is the end, so if you want you can see the node states, we have big matrix but the first one alone is 1 okay.

(Refer Slide Time: 10:14)



The screenshot shows two MATLAB windows side-by-side. The left window displays the following MATLAB code for a Successive Cancellation Decoder:

```
function [y, y_hat] = SCD(aw, h, noise, nbit)
% SCD decoder
% L = nbit+1, M = 2^nbit
% w = received, Mx1
% x0 = zeros(1,M) - node vector
% x1 = zeros(1,PW*L) - channel state vector
%
% f = f(a,b,c) = 1/(2*(a+b)) * (1/(2^c) * b)^(a-1) * (1/(2^c) * a)^(b-1) * b^(a-1)
% g = f(a,b,c) = 1/(2^c) * a! * b! / g function
%
% Ld(i,t) = r_t Modded of root
%
% nbit = Nr. of bits to extract at root
% done = fr blocker has finished or not
% While done == 0 then decode till all bits are decoded
% If not
%   If depth == n
%     If any(f==true) then node from
%       upcap(1, node1) = 0
%     else
%       If k*4+1, node1] >= 0
%         upcap(1, node1) = 0
%       else
%         upcap(1, node1) = 1
%       end
%     end
%   if node == (N-1)
%     done = 1;
%   else
%     node = floor(node/3); depth = depth - 1
%   end
% else
%
```

The right window shows the MATLAB command window with the following output:

```
Octave 3.6.3 (R2012a) for Java (64-bit Java 1.7.0_09-b03) (Ubuntu 12.04.2 LTS)
Type help, doc, or ? for help, describe, or ?function for details.

>> aw = randn(1,100);
>> h = randn(1,100);
>> noise = 0.01;
>> nbit = 10;
>> [y, y_hat] = SCD(aw, h, noise, nbit)

Columns 1 through 3

    -1.7438    2.7327   -0.5906
    -1.0356    -0.5906
    -1.1157    0.9299
    -1.7952   -0.2302
    -0.4310   -1.4814
    -1.4048    1.9208
    -1.7418   -0.2002
    0.5906    1.0356
    -0.5904   -1.1157
    0.9299

Columns 4 through 6

    -0.1593
    -0.0161
    0.1593

>> nbit

nbit =
```

Octave 3.6.3 (R2012a) for Java (64-bit Java 1.7.0_09-b03) (Ubuntu 12.04.2 LTS)
Type help, doc, or ? for help, describe, or ?function for details.

>> aw = randn(1,100);
>> h = randn(1,100);
>> noise = 0.01;
>> nbit = 10;
>> [y, y_hat] = SCD(aw, h, noise, nbit)

Columns 1 through 13

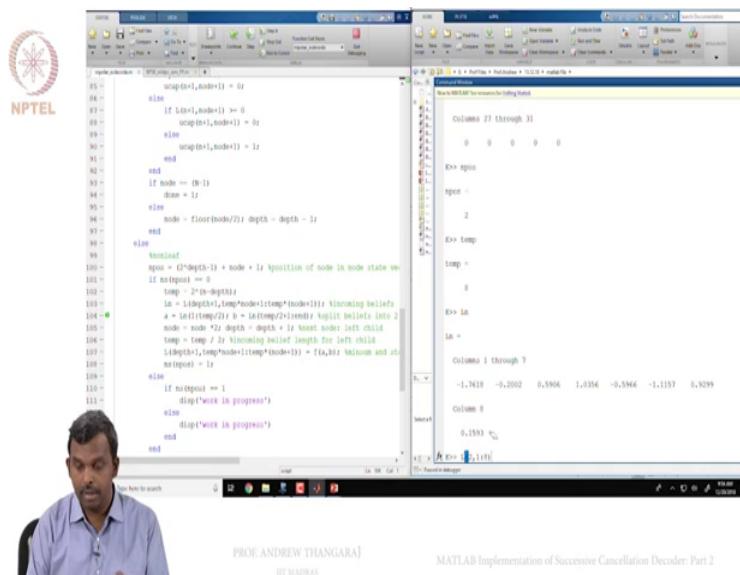
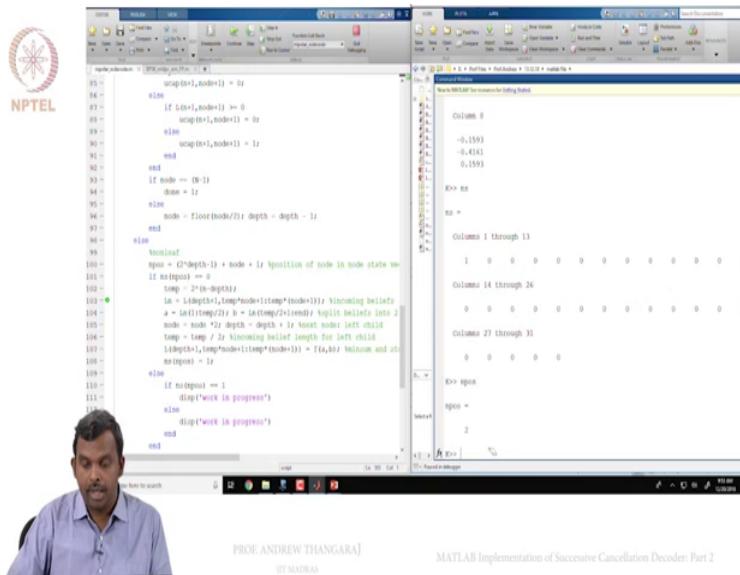
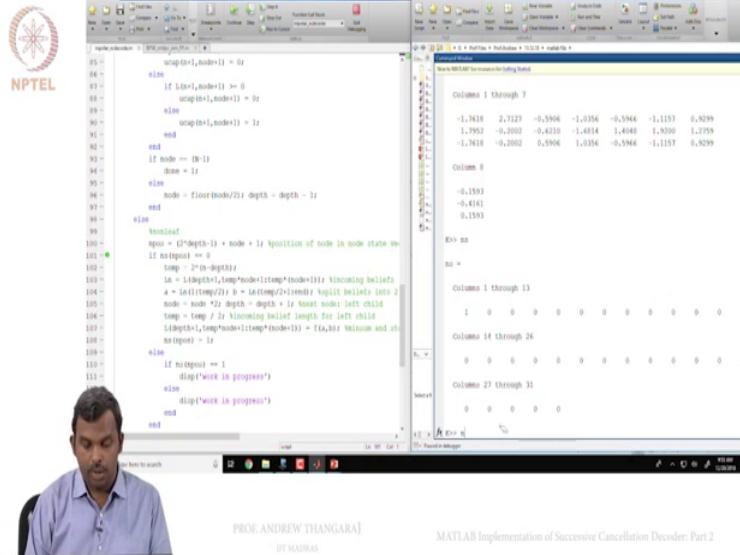
 1 0 0 0 0 0 0 0 0 0 0 0 0

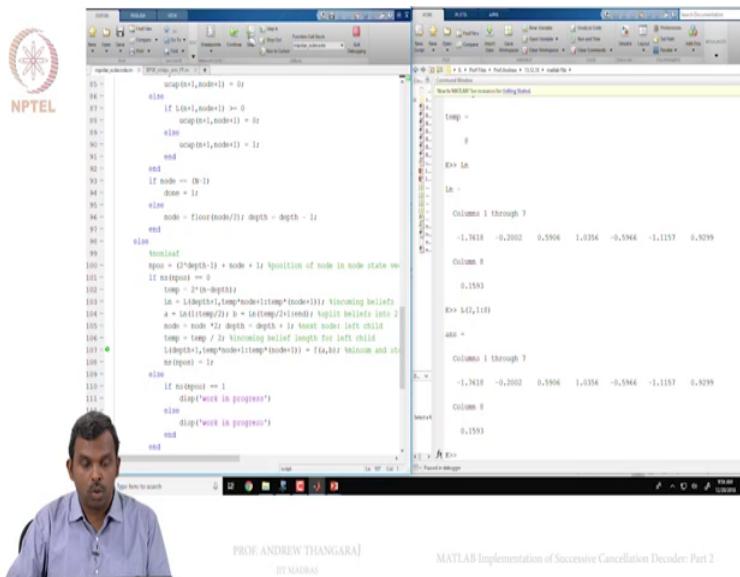
Columns 14 through 24

 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 25 through 31

 0 0 0 0 0 0





The screenshot shows a MATLAB environment. On the left is the code for the Successive Cancellation Decoder:

```

45 - % SCD decoder
46 - %
47 - % Input: Node state vector
48 - % Output: Node state vector
49 - %
50 - % N = number of bits in node state vector
51 - % L = number of bits in channel noise
52 - % Rn = noiseless channel noise vector
53 - %
54 - % f = f(x,y) = (1-2^(-y))x + (1-2^(-y))y * min(|x|,|y|) Minsum function
55 - % g = f(x,y) + 2(1-2^(-y))x * lg lg Function
56 - %
57 - % Node = 1 if node is root
58 - %
59 - % depth = depth of node
60 - %
61 - % done = 1 if node has finished or not
62 - %
63 - % while done == 0 Viterbi till all bits are decoded
64 - %
65 - % if depth >= N
66 - % | if depth == N
67 - % | | if apg(n+1, node+1) >= node frozen
68 - % | | | apg(n+1, node+1) = 0
69 - % | | |
70 - % | | | if L(x(n+1, node+1)) >= 0
71 - % | | | | apg(n+1, node+1) = 1
72 - % | | |
73 - % | | | if apg(n+1, node+1) < 0
74 - % | | | | apg(n+1, node+1) = 0
75 - %
76 - % | end
77 - %
78 - % end
79 - %
80 - % if node == (N-1)
81 - % | done = 1
82 - %
83 - % | node = floor(node/2) depth = depth - 1
84 - %
85 - end

```

On the right, the command window displays the results of the execution:

```

L =

```

Column 1 through 7
-1,7618
-0,2002
0,5906
1,0356
-0,5944
-1,1157
0,9299

Column 8
0,1593


```

done =

```

Column 1 through 7
-1,7618
-0,2002
0,5906
1,0356
-0,5944
-1,1157
0,9299

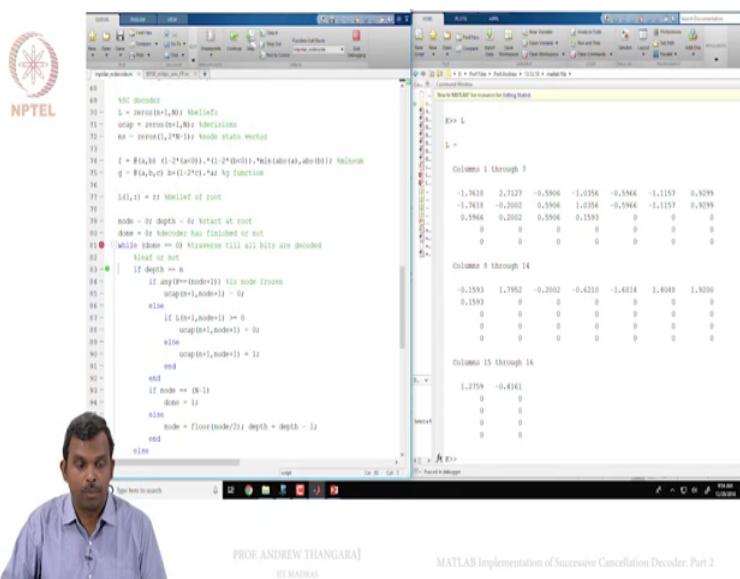
Column 8
0,1593

PROF. ANDREW THANGARA
DT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 2

So the next one, in the next step we are again you do not have this but then if you look at N Poss, N Poss will be 2 I think okay, so that is the correct N Poss, the second position and then again it will be 0 okay, so again it will do this, this will work out hopefully correctly temp will be 8 okay and then LN will pulled out correctly the values that we had, you can see L of 2, 1 colon 8 and agrees with what it pulled out and then you do the same thing as before A, B and then you go to the next node, temp become 4 and then you do L okay and then you set it as N Poss.

(Refer Slide Time: 11:01)



The screenshot shows a MATLAB environment. On the left is the code for the Successive Cancellation Decoder:

```

45 - % SCD decoder
46 - %
47 - % Input: Node state vector
48 - % Output: Node state vector
49 - %
50 - % N = number of bits in node state vector
51 - % L = number of bits in channel noise
52 - % Rn = noiseless channel noise vector
53 - %
54 - % f = f(x,y) = (1-2^(-y))x + (1-2^(-y))y * min(|x|,|y|) Minsum function
55 - % g = f(x,y) + 2(1-2^(-y))x * lg lg Function
56 - %
57 - % Node = 1 if node is root
58 - %
59 - % depth = depth of node
60 - %
61 - % done = 1 if node has finished or not
62 - %
63 - % while done == 0 Viterbi till all bits are decoded
64 - %
65 - % if depth >= N
66 - %
67 - % | if depth == N
68 - % | | if apg(n+1, node+1) >= node frozen
69 - % | | | apg(n+1, node+1) = 0
70 - % | | |
71 - % | | | if L(x(n+1, node+1)) >= 0
72 - % | | | | apg(n+1, node+1) = 1
73 - % | | |
74 - % | | | if apg(n+1, node+1) < 0
75 - % | | | | apg(n+1, node+1) = 0
76 - %
77 - % | end
78 - %
79 - % end
80 - %
81 - % if node == (N-1)
82 - % | done = 1
83 - %
84 - % | node = floor(node/2) depth = depth - 1
85 - %
86 - end

```

On the right, the command window displays the results of the execution:

```

L =

```

Column 1 through 7
-1,7618
2,1227
-0,5906
1,0356
-0,5944
-1,1157
0,9299

Column 8

Column 9 through 14

Column 15 through 14

PROF. ANDREW THANGARA
DT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 2

So if you now you get L will see this big matrix, initially there were 16 values then you have 8 values then you have 4 values okay and then this 4 values you will see are the minsum of this two and you will proceed like this, so you can go up to the leaf node actually.

(Refer Slide Time: 11:19)

PROF. ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 2

```

45 %
46 % SNC decoder
47 %
48 % L = zeros(N+1,M); Nbeliefs
49 % ucap = zeros(N+1,M); Ndecisions
50 % Rn = zeros(1,2^M-1); Node state vector
51 %
52 %
53 % I = R(a,b) 0.274(a+b)+0.726(b-a), 76.1(a+b), 14.0(b)
54 % Q = R(a,b,c) 3+(1-2)c, 4+2c, kg function
55 %
56 L(1,:)=r* Nbeliefs;
57 %
58 Node=0; depth=0; % start at root
59 % Node = r* Vector of N beliefs or not
60 % While there == 0) traverse till all bits are decoded
61 % Node or not
62 % If depth == n
63 % If any(i==Node(i)) via node frozen
64 % ucap(i+1,Node(i)) = 0
65 % else
66 % If i<(i+1,Node(i)) >= 0
67 % ucap(i+1,Node(i)) = 0
68 % else
69 % ucap(i+1,Node(i)) = 1
70 %
71 end
72 %
73 if Node == (N-1)
74 Node = 1;
75 else
76 Node = floor(Node/2);
77 depth = depth - 1;
78 end
79 %
80 if Node == 0
81 done = 1;
82 end
83 %
84 if Node == (N-1)
85 done = 1;
86 end
87 %
88 if Node == 0
89 end
90 %
91 if Node == (N-1)
92 done = 1;
93 end
94 %
95 if Node == 0
96 done = 1;
97 end
98 %
99 if Node == 0
100 spot = (2^(depth-1)) * Node + 1; % position of Node in node state vector
101 if not(spot) == 0
102 temp = 1/(2^(depth-1));
103 L = L(depth,:);temp*node*item*spot); % incoming beliefs
104 % L = L(depth,:);temp*node*item*spot); % incoming beliefs
105 % Node = Node/2; depth = depth + 1; Next node sets only 1
106 % temp = temp / 2; % incoming belief length for last child
107 % L(depth,:);temp*node*item*spot); = ffa,bz minimum and above
108 % spotposition = 1;
109 %
110 if not(spot) == 1
111 disp('work in progress')
112 else
113 disp('work in progress')
114 end
115 end

```

PROF. ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 2

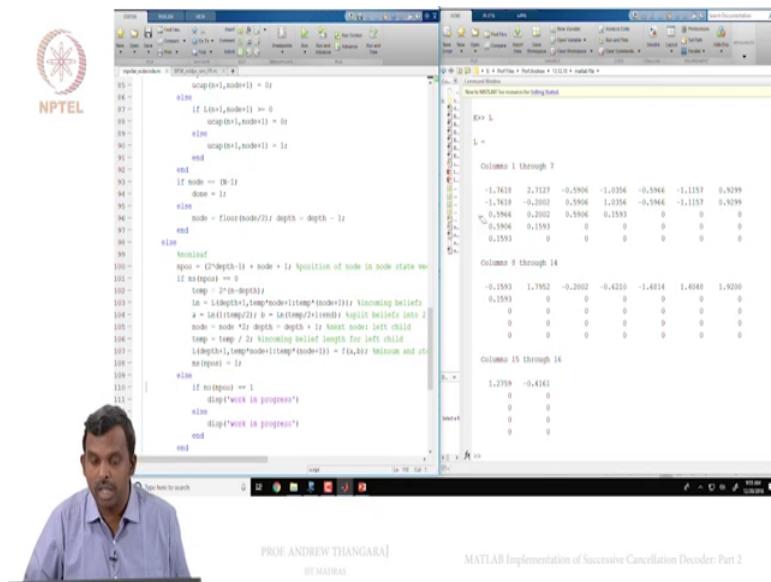
```

45 %
46 % SNC decoder
47 %
48 % L = zeros(N+1,M); Nbeliefs
49 % ucap = zeros(N+1,M); Ndecisions
50 % Rn = zeros(1,2^M-1); Node state vector
51 %
52 %
53 % I = R(a,b) 0.274(a+b)+0.726(b-a), 76.1(a+b), 14.0(b)
54 % Q = R(a,b,c) 3+(1-2)c, 4+2c, kg function
55 %
56 L(1,:)=r* Nbeliefs;
57 %
58 Node=0; depth=0; % start at root
59 % Node = r* Vector of N beliefs or not
60 % While there == 0) traverse till all bits are decoded
61 % Node or not
62 % If depth == n
63 % If any(i==Node(i)) via node frozen
64 % ucap(i+1,Node(i)) = 0
65 % else
66 % If i<(i+1,Node(i)) >= 0
67 % ucap(i+1,Node(i)) = 0
68 % else
69 % ucap(i+1,Node(i)) = 1
70 %
71 end
72 %
73 if Node == (N-1)
74 Node = 1;
75 else
76 Node = floor(Node/2);
77 depth = depth - 1;
78 end
79 %
80 if Node == 0
81 done = 1;
82 end
83 %
84 if Node == (N-1)
85 done = 1;
86 end
87 %
88 if Node == 0
89 end
90 %
91 if Node == (N-1)
92 done = 1;
93 end
94 %
95 if Node == 0
96 done = 1;
97 end
98 %
99 if Node == 0
100 spot = (2^(depth-1)) * Node + 1; % position of Node in node state vector
101 if not(spot) == 0
102 temp = 1/(2^(depth-1));
103 L = L(depth,:);temp*node*item*spot); % incoming beliefs
104 % L = L(depth,:);temp*node*item*spot); % incoming beliefs
105 % Node = Node/2; depth = depth + 1; Next node sets only 1
106 % temp = temp / 2; % incoming belief length for last child
107 % L(depth,:);temp*node*item*spot); = ffa,bz minimum and above
108 % spotposition = 1;
109 %
110 if not(spot) == 1
111 disp('work in progress')
112 else
113 disp('work in progress')
114 end
115 end

```

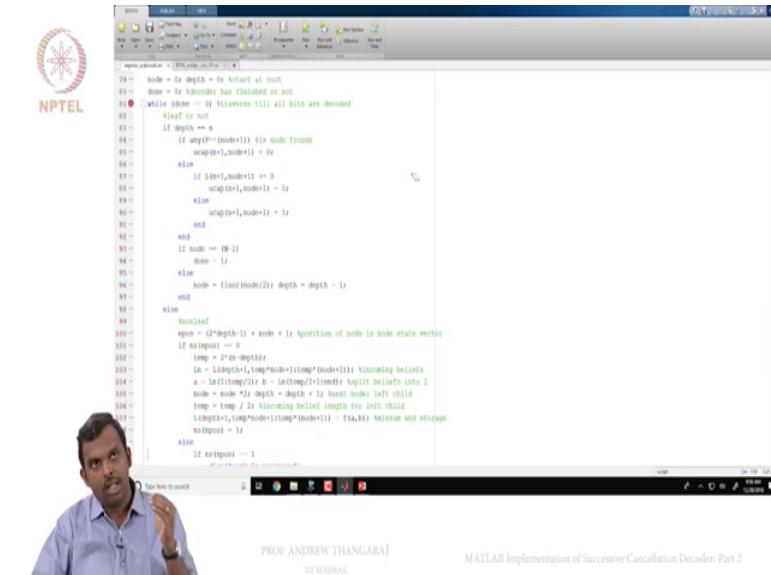
So you go to the next step again the same process continues okay, see L okay come up to this two, now are at the depth of 3, so we are almost at the leaf node, then we can do one more we are ready to leaf node now okay, so you can look at L, you see this triangular sort of structure right 16, 8, 4, 2, 1 and this is your belief for the U1 okay but interestingly U1 is I think frozen, so you if you have depth N it is frozen and then you totally ignored it but still it is okay and node is not N minus 1 and then you go to the next node, node is still 0 you have depth 1.

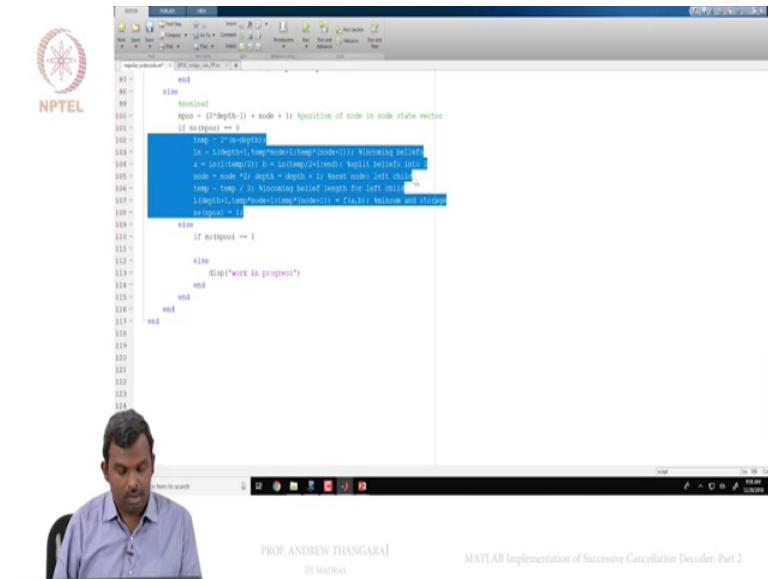
(Refer Slide Time: 12:14)



So you have come up to the next one depth higher okay and here if you do, you know it is not a leaf node okay, you got to the right position but now NS of N Poss will be 1 okay and we have not written code for it, so let us start writing for that okay, so hopefully you saw how the decoder was working, I would give you an idea of how this thing works, how the traversal happens.

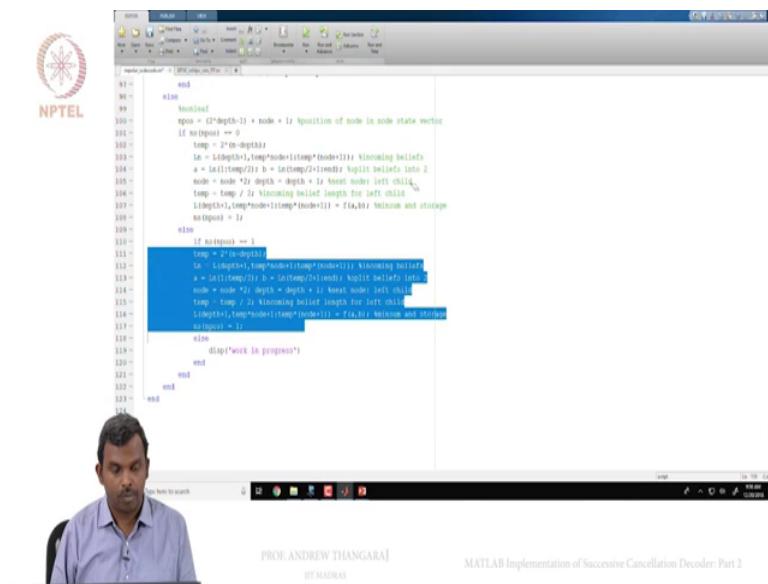
(Refer Slide Time: 12:42)

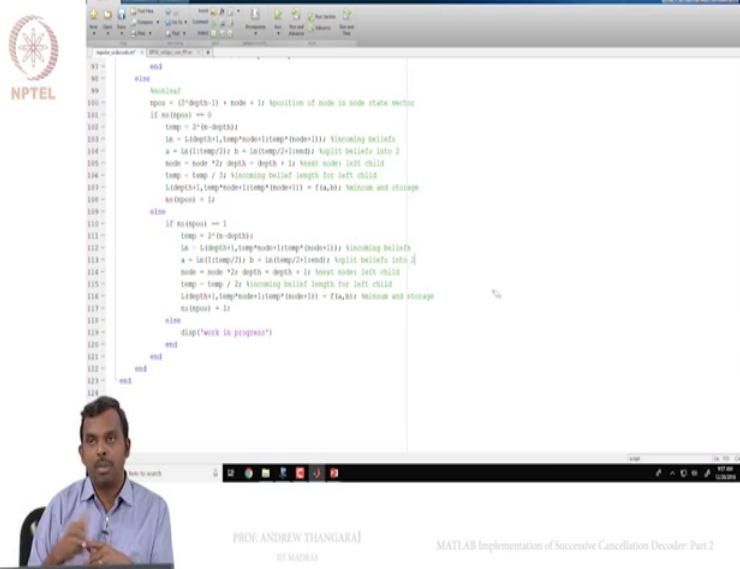




So keep that in mind, so it goes left left left left and then it made a decision and it wanted to go right but we have not written code for going right so let us try and do that now, so once you have figure out how to write for left, the Right is also not very difficult to write down, so I know this, so I will again start the same sort of logic, I need the incoming ones, I need the split okay, I need to know the next node, I need to know the next temp except for this everything else will be the same.

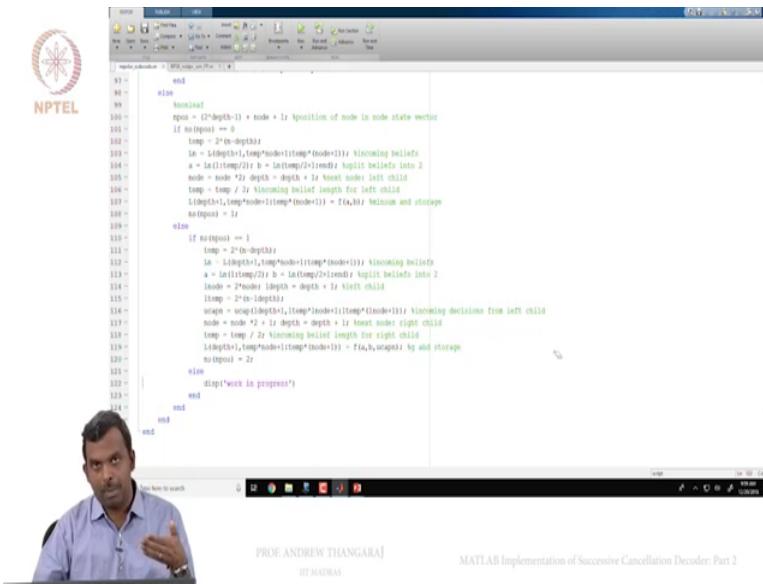
(Refer Slide Time: 13:22)





So I can do a cut and paste here and then modify okay, so you will see there are few changes we have to make here because the state is different, you also need like LN, you also need the, so we got the incoming beliefs, split and all that that is good but we also need the incoming message right, so I am in state 1 okay, so if I am in state 1 it means my left child is giving me something okay.

(Refer Slide Time: 14:00)



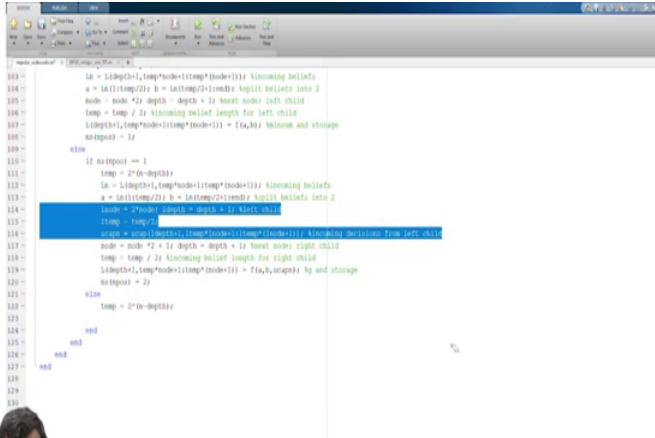
So you have to go to the left child and pick up the U cap from the left child okay, so that is what is important, the left child is U cap is what you have to pick up okay, so what is left child you are at the node and depth, so the left child is 2 into node and the left depth is depth plus 1 right, this is your left child coordinates so to speak okay and then once you do that you can have an L temp which is 2 times 2 power N minus L depth okay and then you can pick up U

cap of the node okay the incoming message from the left child which will be U cap of L depth plus 1, L temp into L node plus 1 colon L temp into L node plus 1 okay.

So this is incoming decisions from left child okay, so hopefully this is clear to you remember when you do the G function, you not only need the incoming beliefs you also need the incoming decisions from the left child okay, so you have to compute who your left child is, go access the corresponding storage for U cap then pull out the storage okay, so for some reason it is warning me that he did not do something correctly I think I did not close the brackets okay, so that is it.

So once you do this, then you go to the next node, now next node has to be the right child okay, the right child is the node into node 2 plus 1 temp still goes down by 2 okay and then you go to the right child, so know you can do A B and then U cap okay, so this is G and storage and now you can make the NS stoop is that okay, so maybe little bit inefficient but I think if we managed to the right thing as well.

(Refer Slide Time: 16:37)



The screenshot shows a MATLAB script window titled 'main.m'. The code implements a Successive Cancellation Decoder for a convolutional channel. It uses a belief propagation algorithm to update belief states for each node in the trellis. The code includes comments explaining the logic for left and right children, belief length calculations, and storage management. A specific line of code is highlighted in blue: 'a = L*temp*2^i; b = L*(temp*2^i*node);' This line is described in the tooltip as 'Assign beliefs into 2 nodes - node * node * depth = depth * 2 * i. Assign node to left child'. The MATLAB interface at the bottom shows standard icons for file operations, and the taskbar indicates other open applications like a browser and a file explorer.

```
131 L = L*temp*2^i,temp=node*itemp*(node+1)) visonning beliefs
132 a = L*(temp*2^i); b = L*(temp*2^i*node); kuglit beliefs into 2
133 node = node * 2^i depth = depth + 1;Assign node to left child
134 temp = temp * 2^i visonning belief length for left child
135 itemp*2^i,temp=node*itemp*(node+1)) = f(a,b);Assign and storage
136 na(node) = 1;
137
138 else
139 if na(node) == 1
140 temp = 2^(n-depth);
141 L = L*temp*2^i,temp=node*itemp*(node+1)); visonning beliefs
142 a = L*(temp*2^i); b = L*(temp*2^i*node); kuglit beliefs into 2
143 node = 2^node; depth = depth + 1;Assign child
144 itemp = temp*2^i,temp=node*itemp*(node+1)); visonning beliefs from left child
145 node = node * 2^i depth = depth + 1;Assign node right child
146 temp = temp * 2^i visonning belief length for right child
147 itemp*2^i,temp=node*itemp*(node+1)) = f(a,b);Assign and storage
148 na(node) = 2;
149
150 else
151 temp = 2^(n-depth);
152 end
153 end
154 end
155 end
156 end
157 end
158 end
159 end
160 end
```



NPTEL

```
130 %
131 %   temp = Ldepths1, temp*node+ltemp*(node+1)) Viterbi beliefs
132 %   a = Lc(ltemp/2), b = Lc(ltemp/2) nodeDr viterb beliefs into 2
133 %   node = node*2+depth - depth + 1) next node left child
134 %   temp = temp / 2r Viterbi belief length for left child
135 %   Ldepths1, temp*node+ltemp*(node+1)) = f(x,y) balloon and storage
136 %   nsdepth = 12
137 %
138 % else
139 % if nsoutput == 1
140 %   temp = 2^t*depth32
141 %   Lc = Ldepths1, temp*node+ltemp*(node+1)) Viterbi beliefs
142 %   a = Lc(ltemp/2), b = Lc(ltemp/2) nodeDr viterb beliefs into 2
143 %   node = 2*node Dr depth = depth + 1) next node left child
144 %   lnode = 2*node Dr depth = depth + 1) next child
145 %   ltemp = temp/2
146 %   wtemp = wtemp + Ldepths1, temp*(node+ltemp*(node+1))) Viterbi decisions from left child
147 %   node = node * 2 + 1r depth = depth + 1r honest nodes right child
148 %   temp = temp / 2r Viterbi belief length for right child
149 %   Ldepths1, temp*node+ltemp*(node+1)) = f(x,y,scap1) by and storage
150 %   nsdepth = 2r
151 %
152 % else
153 %   temp = 2^(t-depth32)
154 %   node = 2*node Dr depth = depth + 1r left child
155 %   Lc = Ldepths1, temp*node+ltemp*(node+1)) Viterbi beliefs from left child
156 %   wtemp = wtemp + Ldepths1, temp*(node+ltemp*(node+1))) Viterbi decisions from left child  $\kappa_0$ 
157 %
158 % end
159 %
160 % end
161 %
162 % end
```



PROF. ANDREW THANGARA]

DT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 2



NPTEL

```
130 %
131 %   temp = Ldepths1, temp*node+ltemp*(node+1)) Viterbi beliefs
132 %   a = Lc(ltemp/2), b = Lc(ltemp/2) nodeDr viterb beliefs into 2
133 %   node = node*2+depth - depth + 1) next node left child
134 %   temp = temp / 2r Viterbi belief length for left child
135 %   Ldepths1, temp*node+ltemp*(node+1)) = f(x,y) balloon and storage
136 %   nsdepth = 12
137 %
138 % else
139 % if nsoutput == 1
140 %   temp = 2^t*depth32
141 %   Lc = Ldepths1, temp*node+ltemp*(node+1)) Viterbi beliefs
142 %   a = Lc(ltemp/2), b = Lc(ltemp/2) nodeDr viterb beliefs into 2
143 %   node = 2*node Dr depth = depth + 1) next node left child
144 %   lnode = 2*node Dr depth = depth + 1) next child
145 %   ltemp = temp/2
146 %   wtemp = wtemp + Ldepths1, temp*(node+ltemp*(node+1))) Viterbi decisions from left child
147 %   node = node * 2 + 1r depth = depth + 1r honest nodes right child
148 %   temp = temp / 2r Viterbi belief length for right child
149 %   Ldepths1, temp*node+ltemp*(node+1)) = f(x,y,scap1) by and storage
150 %   nsdepth = 2r
151 %
152 % else
153 %   temp = 2^(t-depth32)
154 %   node = 2*node Dr depth = depth + 1r left child
155 %   Lc = Ldepths1, temp*node+ltemp*(node+1)) Viterbi beliefs from left child
156 %   wtemp = wtemp + Ldepths1, temp*(node+ltemp*(node+1))) Viterbi decisions from left child  $\kappa_0$ 
157 %
158 % end
159 %
160 % end
161 %
162 % end
```



PROF. ANDREW THANGARA]

DT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 2



NPTEL

```
130 %
131 %   temp = Ldepths1, temp*node+ltemp*(node+1)) Viterbi beliefs
132 %   a = Lc(ltemp/2), b = Lc(ltemp/2) nodeDr viterb beliefs into 2
133 %   node = node*2+depth - depth + 1) next node left child
134 %   temp = temp / 2r Viterbi belief length for left child
135 %   Ldepths1, temp*node+ltemp*(node+1)) = f(x,y) balloon and storage
136 %   nsdepth = 12
137 %
138 % else
139 % if nsoutput == 1
140 %   temp = 2^t*depth32
141 %   Lc = Ldepths1, temp*node+ltemp*(node+1)) Viterbi beliefs
142 %   a = Lc(ltemp/2), b = Lc(ltemp/2) nodeDr viterb beliefs into 2
143 %   node = 2*node Dr depth = depth + 1) next node left child
144 %   lnode = 2*node Dr depth = depth + 1) next child
145 %   ltemp = temp/2
146 %   wtemp = wtemp + Ldepths1, temp*(node+ltemp*(node+1))) Viterbi decisions from left child
147 %   node = node * 2 + 1r depth = depth + 1r honest nodes right child
148 %   temp = temp / 2r Viterbi belief length for right child
149 %   Ldepths1, temp*node+ltemp*(node+1)) = f(x,y,scap1) by and storage
150 %   nsdepth = 2r
151 %
152 % else
153 %   temp = 2^(t-depth32)
154 %   node = 2*node Dr depth = depth + 1r left child
155 %   Lc = Ldepths1, temp*node+ltemp*(node+1)) Viterbi beliefs from left child
156 %   wtemp = wtemp + Ldepths1, temp*(node+ltemp*(node+1))) Viterbi decisions from left child  $\kappa_0$ 
157 %
158 % end
159 %
160 % end
161 %
162 % end
```



PROF. ANDREW THANGARA]

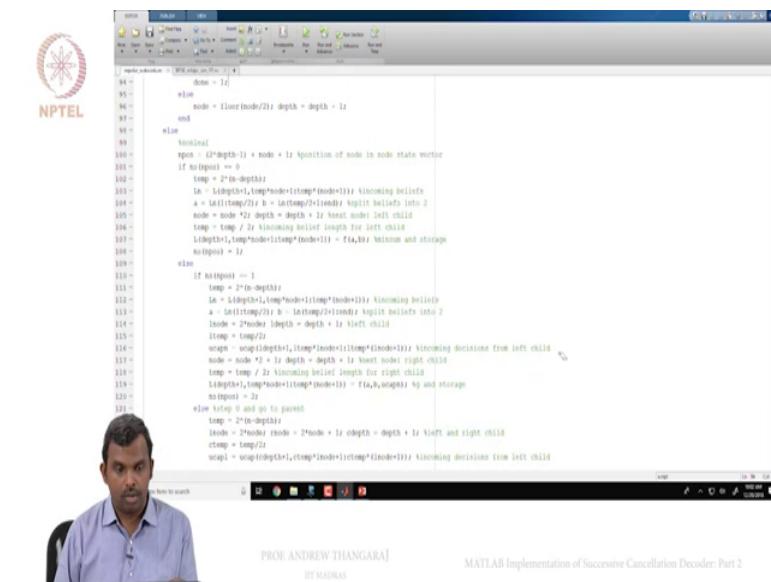
DT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 2

So let us code even the step U which is going back up to the parent, so the so far so good we have not done anythings too wrong, so let us do going up to the parent, going up to the parent is actually also very similar to this in some sense, so in fact here even for L temp I do not think I need, so L depth was just depth plus 1, I can simply say L temp is temp by 2 I think right, so when you go to the left child L temp becomes temp I do just to avoid a little bit of computation that okay.

Alright so this, so let us figure out what to do when NS states is 2 okay, so when state is 2 what you need to do is compute temp okay, so let us compute temp that is okay, you do that and then we do not need A or B but we need the U cap N from the left child okay left node is this, left depth is this okay, so I call it C depth because that is the childs depth and then C temp which is childs temp okay and then U cap left is U cap of C depth plus 1, C temp into L node plus 1, C temp into L node plus 1, so this is incoming decisions from left child.

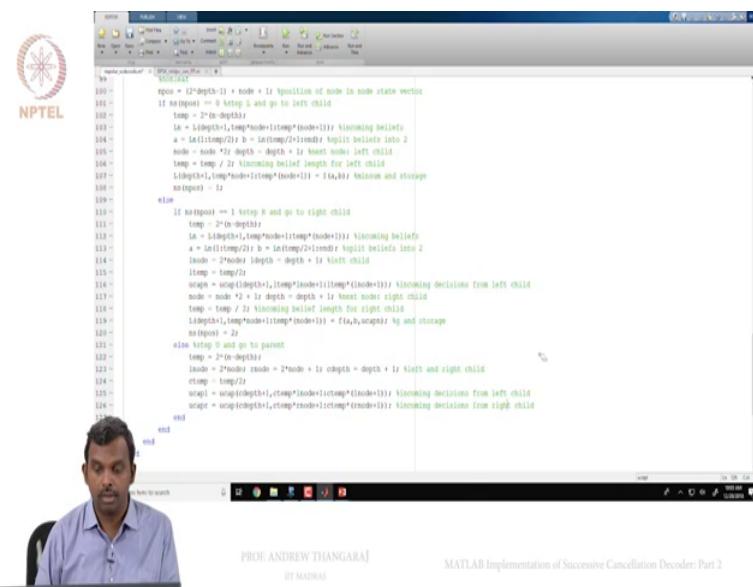
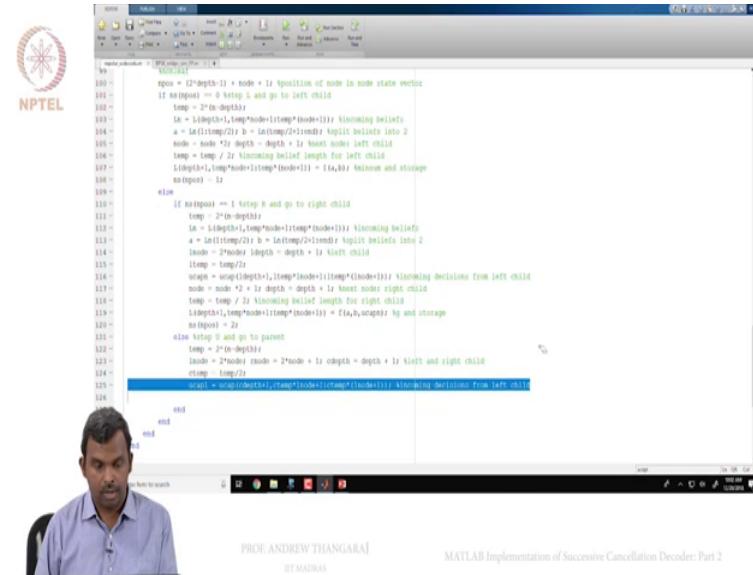
(Refer Slide Time: 18:30)



PROF. ANDREW THANGARA]

MATLAB Implementation of Successive Cancellation Decoder: Part 2

IIIT MADRAS



So I need $U \cap R$ for which I will need R node okay, so I will put R node also right, so you are, so save this is else I need to do step U okay step U and go to parent, remember this, so this is a step L , this is step R okay, so something very similar with $U \cap 2$, $U \cap R$ which is R node incoming decisions from the right child okay.

(Refer Slide Time: 19:50)

So now we are ready to do step 2, step U but for step U you also need the L to come in properly, so that is okay I think we can do that, so just like this assignment you have the assignment for U cap equals, so this is U cap L, we have to do mod 2 right, mod of U cap L plus U cap R, 2 and then U cap R there you go, so I think that step U I think that is correct okay.

So once you do this step you to go back to parent and going back to the parent is done here, so floor of node equals floor of node by 2 and depth equals depth minus 1 then there you go that is it, so the same steps that we did before, we implemented and in fact looks like we are done with the decoder, we have implemented all steps but we have to check whether everything is happening correctly or not, we have not then the debugging part but as far as the basic implementation is concerned we are more or less done.

(Refer Slide Time: 21:26)

```

% MATLAB Implementation of Successive Cancellation Decoder: Part 2
% BY MAURAS

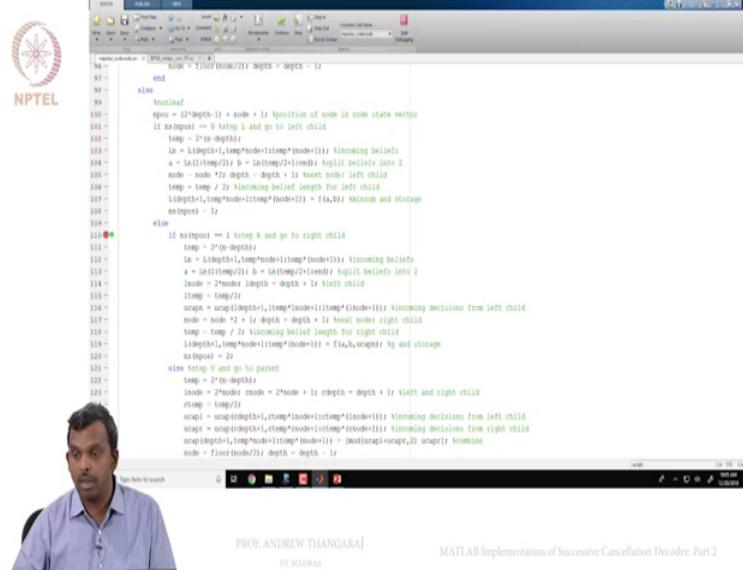
function [node, depth] = SCD(input, noise, h)
    % Initialize variables
    node = 1;
    depth = 1;
    % Main loop
    while depth < 16
        if mod(node, 2) == 0 % If even node
            % Calculate belief length for left child
            temp = 2^(node-1);
            if mod(temp, 2) == 0 % If left child is a leaf
                % Calculate belief length for right child
                temp = temp / 2;
                if mod(temp, 2) == 0 % If right child is a leaf
                    % Calculate belief length for parent
                    temp = temp / 2;
                    % Update node and depth
                    node = floor(node/2);
                    depth = depth - 1;
                else % If right child is not a leaf
                    % Calculate belief length for right child
                    temp = temp / 2;
                    % Update node and depth
                    node = floor(node/2);
                    depth = depth - 1;
                end
            else % If left child is not a leaf
                % Calculate belief length for left child
                temp = temp / 2;
                % Update node and depth
                node = floor(node/2);
                depth = depth - 1;
            end
        else % If odd node
            % Calculate belief length for left child
            temp = 2^(node-1);
            if mod(temp, 2) == 0 % If left child is a leaf
                % Calculate belief length for right child
                temp = temp / 2;
                if mod(temp, 2) == 0 % If right child is a leaf
                    % Calculate belief length for parent
                    temp = temp / 2;
                    % Update node and depth
                    node = floor(node/2);
                    depth = depth - 1;
                else % If right child is not a leaf
                    % Calculate belief length for right child
                    temp = temp / 2;
                    % Update node and depth
                    node = floor(node/2);
                    depth = depth - 1;
                end
            else % If left child is not a leaf
                % Calculate belief length for left child
                temp = temp / 2;
                % Update node and depth
                node = floor(node/2);
                depth = depth - 1;
            end
        end
    end
end

```

PROF. ANDREW THANGARAJ

MATLAB Implementation of Successive Cancellation Decoder: Part 2

BY MAURAS



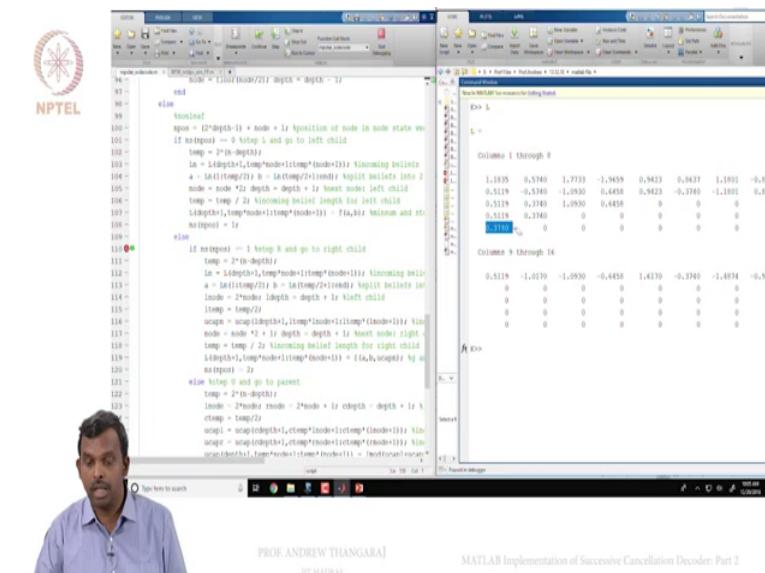
PROF. ANDREW THANGARAJ

DT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 2

So since we already checked the first initial part going up to the first decision I clear this breakpoint and I put a breakpoint here okay, so when you have to come here okay, so when you have to do over here may be okay, so I put a breakpoint here so that I can run this and I have done all the left steps okay and I made a decision and now I have to come back.

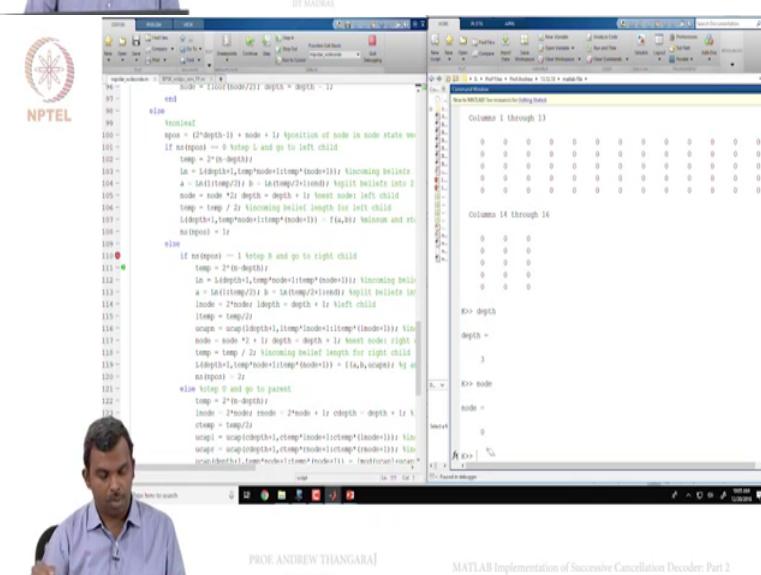
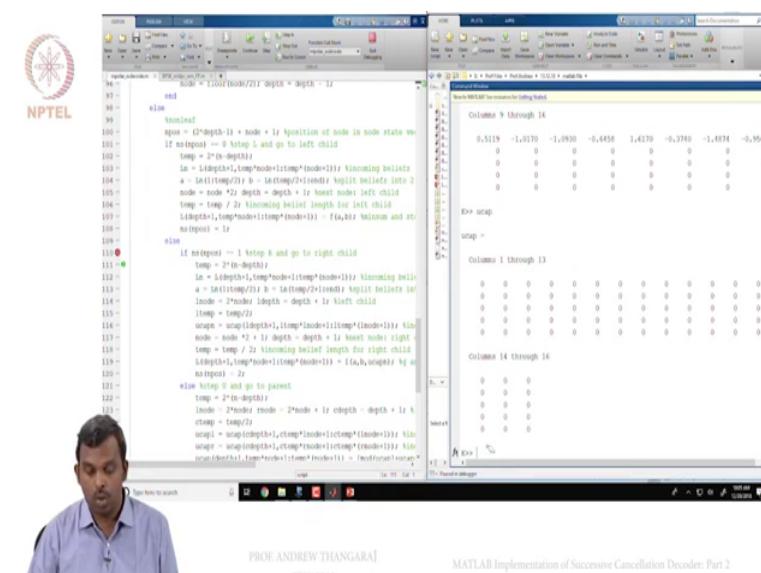
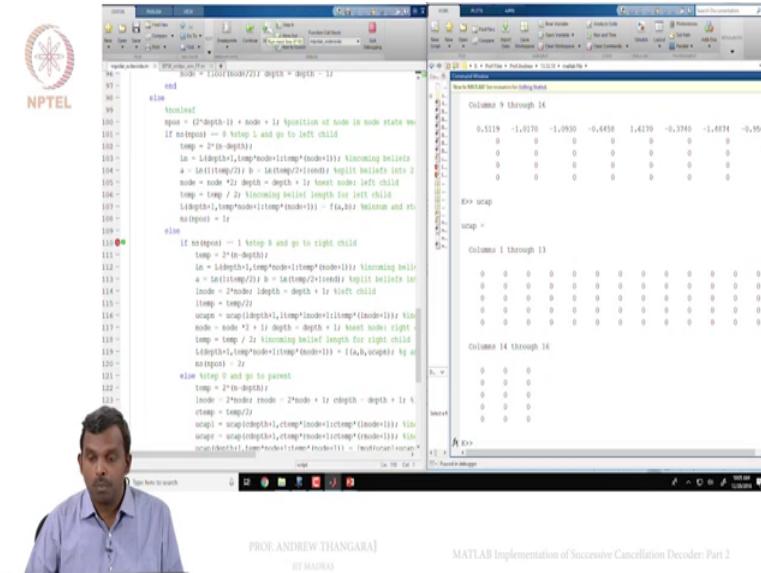
(Refer Slide Time: 21:56)



PROF. ANDREW THANGARAJ

DT MADRAS

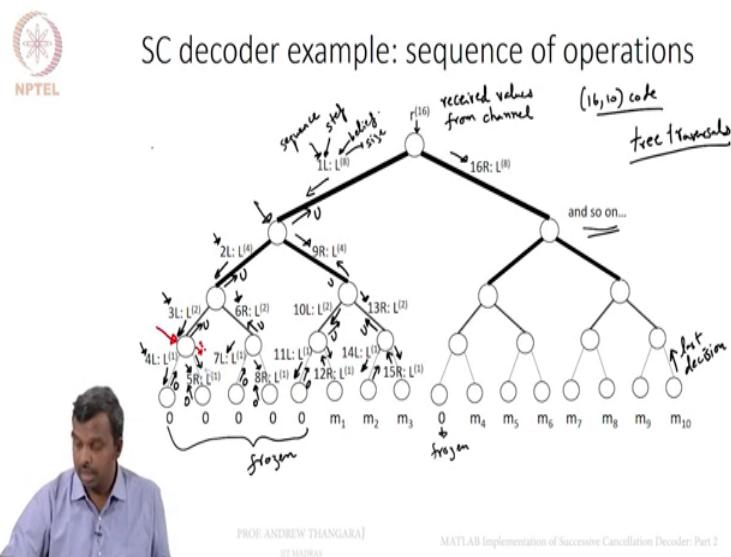
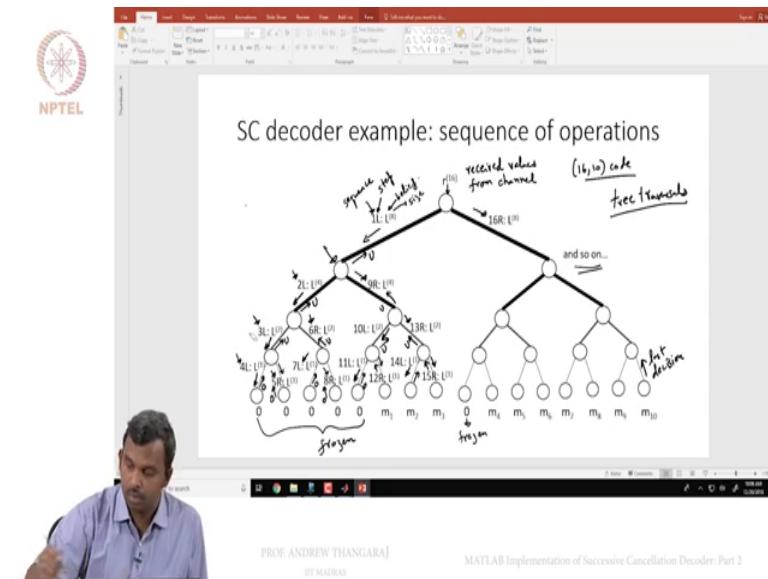
MATLAB Implementation of Successive Cancellation Decoder: Part 2



So just to show you that all that work correctly, let us go back and check the decoder once again, I will clear the screen and I show you the L okay, so you can see the L, let us comeback down to the position here and you can also see the U cap it is just 0 at this point, it is no big B that is where it is and now we are ready to check the right operation okay.

So this point N Poss should be 1 okay, so it is going to, remember what is the node and the depth, so depth is 3 and the node is 0, so where am I now have comeback all the way to the leaf, I have made this decision on the leaf and then I have come back to the previous depth okay, so you remember once again.

(Refer Slide Time: 22:38)



So maybe I should show you go back and show you where we are all on graph, we are somewhere here okay, so here is where we are in the decoder okay, so we went all the way

down made a decision and then we came back and we want to go back here okay, so we need to do that operation and let us check whether the operation is happening correctly or wrong okay.

(Refer Slide Time: 23:07)

PROF. ANDREW THANGARA
BY MAGRAS

```

    % MATLAB Implementation of Successive Cancellation Decoder: Part 2
    % This script implements a successive cancellation decoder for a channel
    % with a specific trellis structure. It uses a loop to process nodes at different depths
    % and updates belief lengths based on received symbols and channel noise.

    % Initialize variables
    node = 1; % Node index
    depth = 1; % Current depth
    L = [0 0 0]; % Belief lengths for three states
    a = 0; % Received symbol
    b = 0; % Received symbol
    temp = 0; % Temporary variable for calculations
    esc = 0; % Escaping flag

    % Loop until all nodes are processed
    while node <= 1000
        if node == 1
            % Initialize first node
            L(1) = 1;
            a = 1;
            b = 1;
            temp = 0;
        else
            % Process previous node
            if esc == 1
                % If escaping, update belief lengths for right child
                L(1) = 1;
                L(2) = 1;
                L(3) = 1;
                esc = 0;
            else
                % If not escaping, update belief lengths for left child
                L(1) = 1;
                L(2) = 1;
                L(3) = 1;
                esc = 1;
            end
        end
        % Print current state
        disp(['node = ', num2str(node), ' depth = ', num2str(depth)])
        % Update node index
        node = 2 * node + 1;
        % Increase depth
        depth = depth + 1;
        % Check for escape condition
        if node == 1000
            esc = 1;
        end
    end

```

PROF. ANDREW THANGARA
BY MAGRAS

```

    % MATLAB Implementation of Successive Cancellation Decoder: Part 2
    % This script implements a successive cancellation decoder for a channel
    % with a specific trellis structure. It uses a loop to process nodes at different depths
    % and updates belief lengths based on received symbols and channel noise.

    % Initialize variables
    node = 1; % Node index
    depth = 1; % Current depth
    L = [0 0 0]; % Belief lengths for three states
    a = 0; % Received symbol
    b = 0; % Received symbol
    temp = 0; % Temporary variable for calculations
    esc = 0; % Escaping flag

    % Loop until all nodes are processed
    while node <= 1000
        if node == 1
            % Initialize first node
            L(1) = 1;
            a = 1;
            b = 1;
            temp = 0;
        else
            % Process previous node
            if esc == 1
                % If escaping, update belief lengths for right child
                L(1) = 1;
                L(2) = 1;
                L(3) = 1;
                esc = 0;
            else
                % If not escaping, update belief lengths for left child
                L(1) = 1;
                L(2) = 1;
                L(3) = 1;
                esc = 1;
            end
        end
        % Print current state
        disp(['node = ', num2str(node), ' depth = ', num2str(depth)])
        % Update node index
        node = 2 * node + 1;
        % Increase depth
        depth = depth + 1;
        % Check for escape condition
        if node == 1000
            esc = 1;
        end
    end

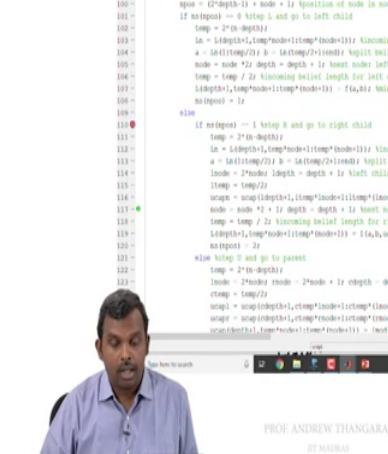
```

PROF ANDREW THANGARA
DT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 2



MATLAB Implementation of Successive Cancellation Decoder: Part 2

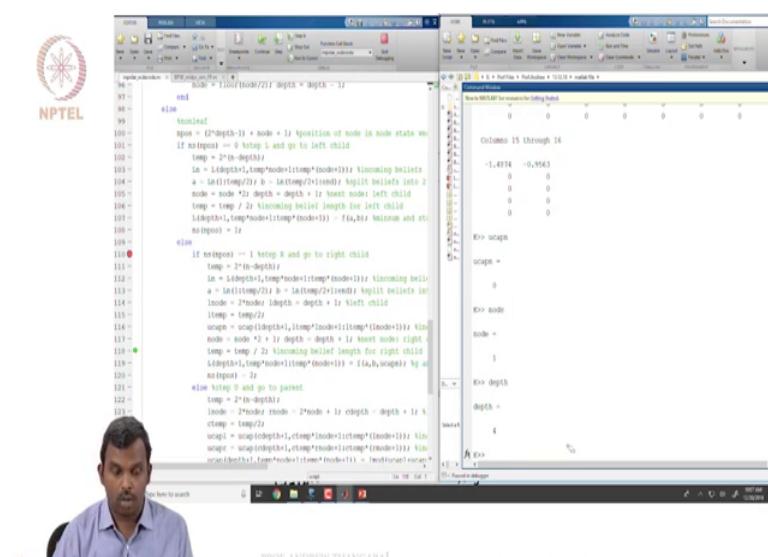
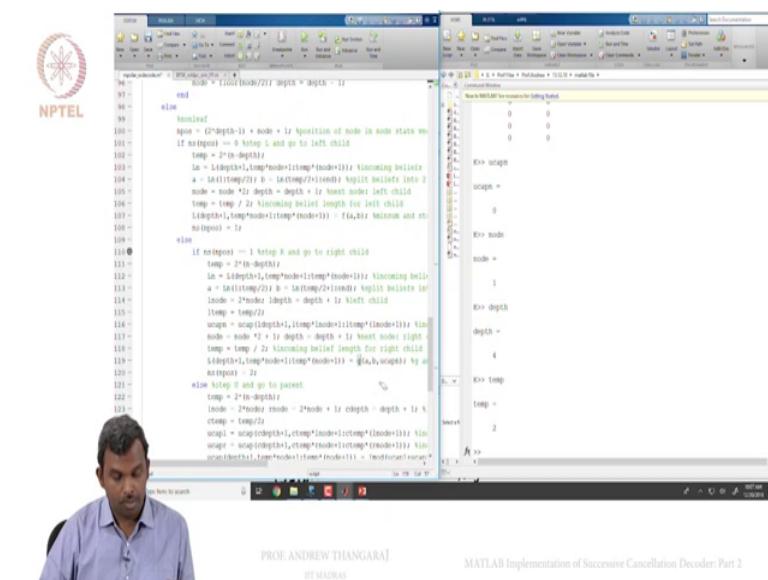
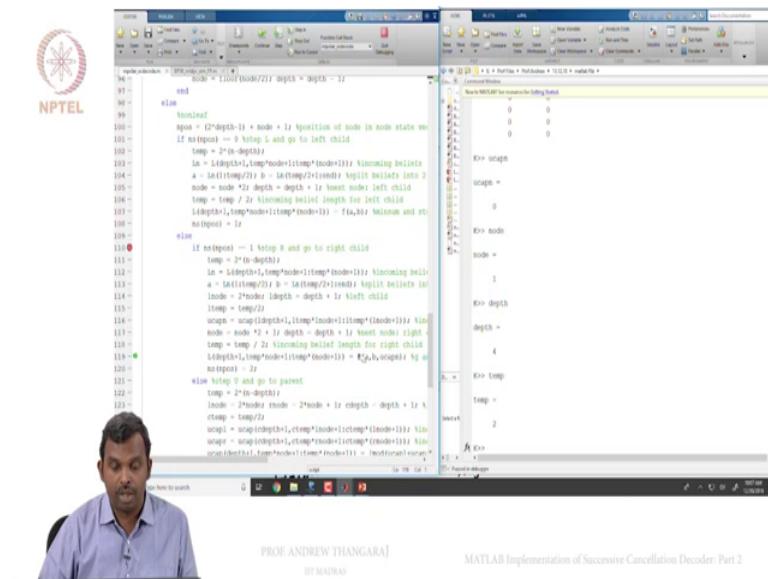


MATLAB Implementation of Successive Cancellation Decoder: Part 2

So that is where we are, step R I am going to write child temp that should be okay, LN that should be okay, A and B that should be okay and then it finds the left child that should be okay and the left temp and then the U cap and it should be okay, so let us check if all those things happen correctly or not okay.

So this is A and B have to be just single values okay and this should just be 0.5149 and 0.3740 okay, so that should be the A and B and U cap N must just be 0 okay, so it should be 0 if it would have picked up the correctly U cap N it assume did that okay, this U cap N it would have picked up okay, so that is okay if you want you can go in and check it but I think it is okay.

(Refer Slide Time: 24:04)



The screenshot shows a MATLAB environment with two windows. On the left is a code editor with MATLAB code for a Successive Cancellation Decoder. On the right is a variable browser showing the state of variables like `usage`, `node`, `depth`, and `temp`. The variable `temp` is currently set to 2.

```

function [usage, node, depth] = SCD(usage, node, depth)
    % Implementation of Successive Cancellation Decoder
    % ...
    % MATLAB Implementation of Successive Cancellation Decoder: Part 2

```

So then it did all that and then is going to the next node okay, so if you look at it now node will be 1 and the depth will be 4 okay, it is gone to the next node, temp probably was 2, so temp needs to go back to 1 okay and then you need to assign it to the correct L right, L is I put at F here it should be G okay, so that is where the array will come, if I run this, this going to be an error, so I will put G here okay, so I made that mistake it should be G, so this will run the correct thing okay.

(Refer Slide Time: 24:40)

The screenshot shows a MATLAB environment with two windows. On the left is a code editor with MATLAB code for a Successive Cancellation Decoder. On the right is a variable browser showing the state of variables like `usage`, `node`, `depth`, and `temp`. The variable `temp` is currently set to 2. A new variable `angular_nodecode` is also visible in the browser.

```

function [usage, node, depth] = SCD(usage, node, depth)
    % Implementation of Successive Cancellation Decoder
    % ...
    % MATLAB Implementation of Successive Cancellation Decoder: Part 2

```



NPTEL

PROF ANDREW THANGARA
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 2



NPTEL

PROF ANDREW THANGARA
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 2

So let us run it once again and quickly step through all the way up to here, things are working correctly and then this assign the L for that kind okay, so let see what happen to the L, this happened correctly or not yes okay, so this must be just the sum of these two and you can see 0.33 is correct okay, so it added these two and you got the sum, that is the L value okay.

(Refer Slide Time: 25:10)

PROF. ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 2

```
111 %  
112 %  
113 %  
114 %  
115 %  
116 %  
117 %  
118 %  
119 %  
120 %  
121 %  
122 %  
123 %  
124 %  
125 %  
126 %  
127 %  
128 %  
129 %  
130 %  
131 %  
132 %  
133 %  
134 %  
135 %  
136 %  
137 %  
138 %  
139 %  
140 %  
141 %  
142 %  
143 %  
144 %  
145 %  
146 %  
147 %  
148 %  
149 %  
150 %  
151 %  
152 %  
153 %  
154 %  
155 %  
156 %  
157 %  
158 %  
159 %  
160 %  
161 %  
162 %  
163 %  
164 %  
165 %  
166 %  
167 %  
168 %  
169 %  
170 %  
171 %  
172 %  
173 %  
174 %  
175 %  
176 %  
177 %  
178 %  
179 %  
180 %  
181 %  
182 %  
183 %  
184 %  
185 %  
186 %  
187 %  
188 %  
189 %  
190 %  
191 %  
192 %  
193 %  
194 %  
195 %  
196 %  
197 %  
198 %  
199 %  
200 %  
201 %  
202 %  
203 %  
204 %  
205 %  
206 %  
207 %  
208 %  
209 %  
210 %  
211 %  
212 %  
213 %  
214 %  
215 %  
216 %  
217 %  
218 %  
219 %  
220 %  
221 %  
222 %  
223 %  
224 %  
225 %  
226 %  
227 %  
228 %  
229 %  
230 %  
231 %  
232 %  
233 %  
234 %  
235 %  
236 %  
237 %  
238 %  
239 %  
240 %  
241 %  
242 %  
243 %  
244 %  
245 %  
246 %  
247 %  
248 %  
249 %  
250 %  
251 %  
252 %  
253 %  
254 %  
255 %  
256 %  
257 %  
258 %  
259 %  
260 %  
261 %  
262 %  
263 %  
264 %  
265 %  
266 %  
267 %  
268 %  
269 %  
270 %  
271 %  
272 %  
273 %  
274 %  
275 %  
276 %  
277 %  
278 %  
279 %  
280 %  
281 %  
282 %  
283 %  
284 %  
285 %  
286 %  
287 %  
288 %  
289 %  
290 %  
291 %  
292 %  
293 %  
294 %  
295 %  
296 %  
297 %  
298 %  
299 %  
300 %  
301 %  
302 %  
303 %  
304 %  
305 %  
306 %  
307 %  
308 %  
309 %  
310 %  
311 %  
312 %  
313 %  
314 %  
315 %  
316 %  
317 %  
318 %  
319 %  
320 %  
321 %  
322 %  
323 %  
324 %  
325 %  
326 %  
327 %  
328 %  
329 %  
330 %  
331 %  
332 %  
333 %  
334 %  
335 %  
336 %  
337 %  
338 %  
339 %  
340 %  
341 %  
342 %  
343 %  
344 %  
345 %  
346 %  
347 %  
348 %  
349 %  
350 %  
351 %  
352 %  
353 %  
354 %  
355 %  
356 %  
357 %  
358 %  
359 %  
360 %  
361 %  
362 %  
363 %  
364 %  
365 %  
366 %  
367 %  
368 %  
369 %  
370 %  
371 %  
372 %  
373 %  
374 %  
375 %  
376 %  
377 %  
378 %  
379 %  
380 %  
381 %  
382 %  
383 %  
384 %  
385 %  
386 %  
387 %  
388 %  
389 %  
390 %  
391 %  
392 %  
393 %  
394 %  
395 %  
396 %  
397 %  
398 %  
399 %  
400 %  
401 %  
402 %  
403 %  
404 %  
405 %  
406 %  
407 %  
408 %  
409 %  
410 %  
411 %  
412 %  
413 %  
414 %  
415 %  
416 %  
417 %  
418 %  
419 %  
420 %  
421 %  
422 %  
423 %  
424 %  
425 %  
426 %  
427 %  
428 %  
429 %  
430 %  
431 %  
432 %  
433 %  
434 %  
435 %  
436 %  
437 %  
438 %  
439 %  
440 %  
441 %  
442 %  
443 %  
444 %  
445 %  
446 %  
447 %  
448 %  
449 %  
450 %  
451 %  
452 %  
453 %  
454 %  
455 %  
456 %  
457 %  
458 %  
459 %  
460 %  
461 %  
462 %  
463 %  
464 %  
465 %  
466 %  
467 %  
468 %  
469 %  
470 %  
471 %  
472 %  
473 %  
474 %  
475 %  
476 %  
477 %  
478 %  
479 %  
480 %  
481 %  
482 %  
483 %  
484 %  
485 %  
486 %  
487 %  
488 %  
489 %  
490 %  
491 %  
492 %  
493 %  
494 %  
495 %  
496 %  
497 %  
498 %  
499 %  
500 %  
501 %  
502 %  
503 %  
504 %  
505 %  
506 %  
507 %  
508 %  
509 %  
510 %  
511 %  
512 %  
513 %  
514 %  
515 %  
516 %  
517 %  
518 %  
519 %  
520 %  
521 %  
522 %  
523 %  
524 %  
525 %  
526 %  
527 %  
528 %  
529 %  
530 %  
531 %  
532 %  
533 %  
534 %  
535 %  
536 %  
537 %  
538 %  
539 %  
540 %  
541 %  
542 %  
543 %  
544 %  
545 %  
546 %  
547 %  
548 %  
549 %  
550 %  
551 %  
552 %  
553 %  
554 %  
555 %  
556 %  
557 %  
558 %  
559 %  
560 %  
561 %  
562 %  
563 %  
564 %  
565 %  
566 %  
567 %  
568 %  
569 %  
570 %  
571 %  
572 %  
573 %  
574 %  
575 %  
576 %  
577 %  
578 %  
579 %  
580 %  
581 %  
582 %  
583 %  
584 %  
585 %  
586 %  
587 %  
588 %  
589 %  
590 %  
591 %  
592 %  
593 %  
594 %  
595 %  
596 %  
597 %  
598 %  
599 %  
600 %  
601 %  
602 %  
603 %  
604 %  
605 %  
606 %  
607 %  
608 %  
609 %  
610 %  
611 %  
612 %  
613 %  
614 %  
615 %  
616 %  
617 %  
618 %  
619 %  
620 %  
621 %  
622 %  
623 %  
624 %  
625 %  
626 %  
627 %  
628 %  
629 %  
630 %  
631 %  
632 %  
633 %  
634 %  
635 %  
636 %  
637 %  
638 %  
639 %  
640 %  
641 %  
642 %  
643 %  
644 %  
645 %  
646 %  
647 %  
648 %  
649 %  
650 %  
651 %  
652 %  
653 %  
654 %  
655 %  
656 %  
657 %  
658 %  
659 %  
660 %  
661 %  
662 %  
663 %  
664 %  
665 %  
666 %  
667 %  
668 %  
669 %  
670 %  
671 %  
672 %  
673 %  
674 %  
675 %  
676 %  
677 %  
678 %  
679 %  
680 %  
681 %  
682 %  
683 %  
684 %  
685 %  
686 %  
687 %  
688 %  
689 %  
690 %  
691 %  
692 %  
693 %  
694 %  
695 %  
696 %  
697 %  
698 %  
699 %  
700 %  
701 %  
702 %  
703 %  
704 %  
705 %  
706 %  
707 %  
708 %  
709 %  
710 %  
711 %  
712 %  
713 %  
714 %  
715 %  
716 %  
717 %  
718 %  
719 %  
720 %  
721 %  
722 %  
723 %  
724 %  
725 %  
726 %  
727 %  
728 %  
729 %  
730 %  
731 %  
732 %  
733 %  
734 %  
735 %  
736 %  
737 %  
738 %  
739 %  
740 %  
741 %  
742 %  
743 %  
744 %  
745 %  
746 %  
747 %  
748 %  
749 %  
750 %  
751 %  
752 %  
753 %  
754 %  
755 %  
756 %  
757 %  
758 %  
759 %  
760 %  
761 %  
762 %  
763 %  
764 %  
765 %  
766 %  
767 %  
768 %  
769 %  
770 %  
771 %  
772 %  
773 %  
774 %  
775 %  
776 %  
777 %  
778 %  
779 %  
780 %  
781 %  
782 %  
783 %  
784 %  
785 %  
786 %  
787 %  
788 %  
789 %  
790 %  
791 %  
792 %  
793 %  
794 %  
795 %  
796 %  
797 %  
798 %  
799 %  
800 %  
801 %  
802 %  
803 %  
804 %  
805 %  
806 %  
807 %  
808 %  
809 %  
810 %  
811 %  
812 %  
813 %  
814 %  
815 %  
816 %  
817 %  
818 %  
819 %  
820 %  
821 %  
822 %  
823 %  
824 %  
825 %  
826 %  
827 %  
828 %  
829 %  
830 %  
831 %  
832 %  
833 %  
834 %  
835 %  
836 %  
837 %  
838 %  
839 %  
840 %  
841 %  
842 %  
843 %  
844 %  
845 %  
846 %  
847 %  
848 %  
849 %  
850 %  
851 %  
852 %  
853 %  
854 %  
855 %  
856 %  
857 %  
858 %  
859 %  
860 %  
861 %  
862 %  
863 %  
864 %  
865 %  
866 %  
867 %  
868 %  
869 %  
870 %  
871 %  
872 %  
873 %  
874 %  
875 %  
876 %  
877 %  
878 %  
879 %  
880 %  
881 %  
882 %  
883 %  
884 %  
885 %  
886 %  
887 %  
888 %  
889 %  
890 %  
891 %  
892 %  
893 %  
894 %  
895 %  
896 %  
897 %  
898 %  
899 %  
900 %  
901 %  
902 %  
903 %  
904 %  
905 %  
906 %  
907 %  
908 %  
909 %  
910 %  
911 %  
912 %  
913 %  
914 %  
915 %  
916 %  
917 %  
918 %  
919 %  
920 %  
921 %  
922 %  
923 %  
924 %  
925 %  
926 %  
927 %  
928 %  
929 %  
930 %  
931 %  
932 %  
933 %  
934 %  
935 %  
936 %  
937 %  
938 %  
939 %  
940 %  
941 %  
942 %  
943 %  
944 %  
945 %  
946 %  
947 %  
948 %  
949 %  
950 %  
951 %  
952 %  
953 %  
954 %  
955 %  
956 %  
957 %  
958 %  
959 %  
960 %  
961 %  
962 %  
963 %  
964 %  
965 %  
966 %  
967 %  
968 %  
969 %  
970 %  
971 %  
972 %  
973 %  
974 %  
975 %  
976 %  
977 %  
978 %  
979 %  
980 %  
981 %  
982 %  
983 %  
984 %  
985 %  
986 %  
987 %  
988 %  
989 %  
990 %  
991 %  
992 %  
993 %  
994 %  
995 %  
996 %  
997 %  
998 %  
999 %  
1000 %
```

PROF. ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 2

So the next step and made the state as two for this and see I am not bothering once the U happens to change the state because when I could change it but I will not come back to it ever again, so it does not matter too much, then we continue and then it will go to decision okay, so it will make a decision, it is frozen again and it is not so it is goes back up in depth to the next one okay.

(Refer Slide Time: 25:37)

```

49 %SC decoder
50
51 L = zeros(8*1,M); M=lelf0;
52 uscp = zeros(8*1,M); M=lelf0;
53 Rn = zeros(1,2^M-1); Node state vector
54
55 f = f(x,M)-2*(a*x(1)+1)*2^(M-1)*min(a(x),abs(b(1))); Minimise
56 Q = f(x,A,B,C)+2*(1-2*x1)*A*x kg function
57
58 L(1,:)=r; %Initial of root
59
60 node = 0; depth = 0; %Start at root
61 done = 0; %Decoder has finished or not
62 while done == 0 %Iterate till all bits are decoded
63     Next or not
64     if depth == n
65         if any(f==node(i)) %is node frozen
66             uscp(i,Node(i)) = 0;
67         else
68             if L(i,Node(i)) > 0
69                 uscp(i,Node(i)) = 0;
70             else
71                 uscp(i+1,Node(i)) = 1;
72             end
73         end
74     end
75     if node == (N-1)
76         done = 1;
77     else
78         node = floor(node/2); depth = depth + 1;
79     end
80 end

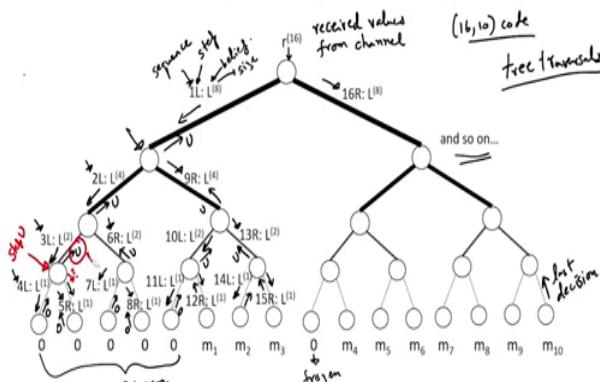
```

PROF. ANDREW THANGARA
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 2.



SC decoder example: sequence of operations



PROF. ANDREW THANGARA
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 2.



```

108 temp = temp/2; %Increasing belief length for left child
109 L(1:temp+1,temp:node+ltemp*(node+1)) = f(x,M); Minimise and store
110 ns(temp) = 1;
111
112 if ns(temp) == 1 %Step 8 and go to right child
113     temp = 2*temp;
114     Le = 2*temp;Ltemp, temp=node+ltemp*(node+1); Increasing belief
115     a = Le+ltemp/2; b = Le+ltemp/2; ltempdir; Right beliefs in
116     lnode = 2*node; length = depth + 1; Left child
117     ltemp = temp/2;
118     uscp = uscp(ltemp+1,ltemp*Mnode+ltemp*(node+1)); Minimise
119     node = node*2 + 1; depth = depth + 1; Next node right
120     temp = temp/2; %Increasing belief length for right child
121     Rr = 2*temp;Rtemp, temp=node+ltemp*(node+1); Increasing belief
122     rnode = 2*node+1; length = depth + 1; Right child
123     rtemp = temp/2;
124     uscp = uscp(rtemp+1,rtemp*Mnode+rtemp*(node+1)); Minimise
125     node = floor(node/2); depth = depth - 1;
126
127 end
128
129 end
130 end
131 end
132 end

```

PROF. ANDREW THANGARA
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 2.

The screenshot shows a MATLAB IDE window. On the left is a code editor with MATLAB code for a Successive Cancellation Decoder. On the right is a variable browser showing the state of variables like depth, node, and temp.

```

108 %
109 %      temp = temp / 2; binarizing belief length for left child
110 %      ldepth1,temp*node+itemp*(node+1)) = f1a,B1 minimum and st
111 %      ntemp = 1;
112 %
113 %      if ntemp == 1 step k and go to right child
114 %          temp = Ldepth1,temp*node+itemp*(node+1)) binarizing beliefs
115 %          a = 16*temp/2; r = 16*temp/2*node; split beliefs into 2
116 %          node = node * 2 + 1; depth = depth + 1; next node's left child
117 %          temp = temp / 2; binarizing belief length for left child
118 %          ldepth1,temp*node+itemp*(node+1)) = f1a,B1 minimum and st
119 %          ntemp = 1;
120 %
121 %      else if ntemp >= 2
122 %          temp = 2*depth;
123 %          lnode = 2*node; rnode = 2*node + 1; depth = depth + 1; r = 1;
124 %          ctemp = temp/2;
125 %          uscap = uscap(ldepth1,temp*node+itemp*(node+1)); lnode
126 %          uscap = uscap(ldepth1,temp*node+itemp*(node+1)); rnode
127 %          uscap(ldepth1,temp*node+itemp*(node+1)) = (node*uscap)+uscap;
128 %
129 %          node = floor(node/2); depth = depth - 1;
130 %
131 %      end
132 %
133 %  end
134 %
135 % end

```

PROF. ANDREW THANGARA
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 2

So now it will come to once again where is it come okay, so you remember it is finished the decision here and it is comeback here, now this node is here to do step U okay, so it is needs to figure out this step U here and it will do it, we will see it does it correctly okay, so depth is not N now, so N Poss it calculates and step, it is not 0, it is not 1, it will go to 2 okay.

So now it has to do step 2, calculates the depth and then the L temp, L node and C depth, so let just check it out once again, so depth is should be 3, node should be 0, it going to go up right, so and then let us look at L node it will be 0 again and C depth will be 4 okay, so that is okay and then it calculates C temp that is okay, it does that C temp should just be 1 okay, I can check that it is 1 and then it pulls out the U cap both from L node and I am not, this will just be 1 bit okay.

(Refer Slide Time: 27:02)

The screenshot shows a MATLAB IDE window. On the left is a code editor with MATLAB code for a Successive Cancellation Decoder. On the right is a variable browser showing the state of variables like depth, node, and temp.

```

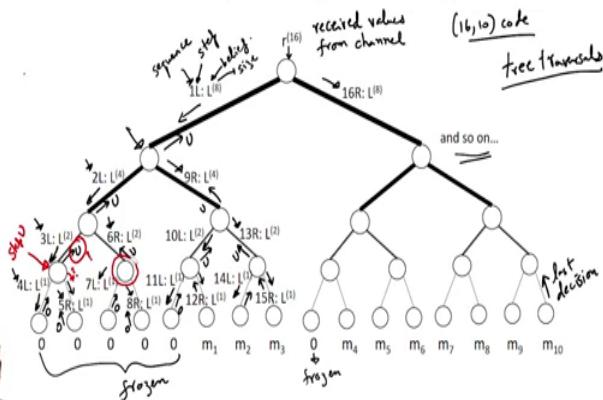
108 %
109 %      if ntemp == 1 step k and go to left child
110 %          temp = 2*depth;
111 %          lnode = 2*node; rnode = 2*node + 1; depth = depth + 1; r = 1;
112 %          a = 16*temp/2; r = 16*temp/2*node; split beliefs into 2
113 %          node = node * 2 + 1; depth = depth + 1; next node's left child
114 %          temp = temp / 2; binarizing belief length for left child
115 %          ldepth1,temp*node+itemp*(node+1)) = f1a,B1 minimum and st
116 %          ntemp = 1;
117 %
118 %      else if ntemp >= 2
119 %          temp = 2*depth;
120 %          lnode = 2*node; rnode = 2*node + 1; depth = depth + 1; r = 1;
121 %          ctemp = temp/2;
122 %          uscap = uscap(ldepth1,temp*node+itemp*(node+1)); lnode
123 %          uscap = uscap(ldepth1,temp*node+itemp*(node+1)); rnode
124 %          uscap(ldepth1,temp*node+itemp*(node+1)) = (node*uscap)+uscap;
125 %
126 %          node = floor(node/2); depth = depth - 1;
127 %
128 %      end
129 %
130 %  end
131 %
132 % end

```

PROF. ANDREW THANGARA
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 2

SC decoder example: sequence of operations



So U cap L and U cap R will both be 1 bit and there will be 0 because they were frozen positions and then you calculate the U cap for going up okay, so if you do this make it again busy ruse it is, so if CU cap I mean everything is going to be 0, so you not going to see much and then you go up to the parent okay, when you went up to the parent you would have gone to depth of 2 and the node would be 0 okay that again to 0 and then after this it will continue, depth is not N and calculates N Poss, this will be actually 1.

So it will do the right calculation, so it is good to check what it did here, so remember this going to go right, node is 1, depth is 3 okay, so what is node 1 step depth 3, so maybe you can see what is node 1 in depth 3, so it is come here and node while it is come here okay, it is come to this guy okay and that come there which is be sure, let me just see the L here yes, so this is fine so this goes to node one in depth 3, yes so I think I went through the L and it just is not pay attention to okay, so that is okay, so that is good.

(Refer Slide Time: 28:46)

The screenshot shows a MATLAB environment with two windows. On the left, a code editor displays MATLAB code for a successive cancellation decoder. On the right, a command window shows several large matrices labeled 'Column 1 through 7', 'Column 8 through 14', and 'Column 15 through 14'. Below the code editor, a video player window shows a video of Prof. Andrew Thangara. At the bottom, a footer bar indicates 'PROF. ANDREW THANGARA' and 'IIT MADRAS'.

```

101 - %if nstop == 0 step 1 and go to left child
102 - temp = 2^(b-depth);
103 - tn = 1depth1, temp*node1temp*(node+1)); %scanning beliefs
104 - a = 1depth1, temp*node1temp*(node+1));
105 - node = node*2^b depth + 1; %next node's left child
106 - temp = temp / 2; %scanning belief length for left child
107 - 1depth1, temp*node1temp*(node+1)) = f1a1a1(node, and, ut,
108 - nstop) = 1;
109 -
110 - else
111 - if nstop == 1 step 2 and go to right child
112 - temp = 2^(b-depth);
113 - tn = 1depth1, temp*node1temp*(node+1)); %scanning beliefs
114 - a = 1depth1, temp*node1temp*(node+1));
115 - node = node*2^b depth + 1; %next node's right child
116 - temp = temp / 2; %scanning belief length for right child
117 - 1depth1, temp*node1temp*(node+1)) = q1a2ucapq1(q1a2,
118 - nstop) = 1;
119 - end
120 - end
121 - end
122 - end
123 - end
124 - end
125 - end
126 - end
127 - end
128 - end
129 - end
130 - end
131 - end
132 - end
133 - end
134 - end
135 - end
136 - end
137 - end
138 - end
139 - end
140 - end
141 - end
142 - end
143 - end
144 - end
145 - end
146 - end
147 - end
148 - end
149 - end
150 - end
151 - end
152 - end
153 - end
154 - end
155 - end
156 - end
157 - end
158 - end
159 - end
160 - end
161 - end
162 - end
163 - end
164 - end
165 - end
166 - end
167 - end
168 - end
169 - end
170 - end
171 - end
172 - end
173 - end
174 - end
175 - end
176 - end
177 - end
178 - end
179 - end
180 - end
181 - end
182 - end
183 - end
184 - end
185 - end
186 - end
187 - end
188 - end
189 - end
190 - end
191 - end
192 - end
193 - end
194 - end
195 - end
196 - end
197 - end
198 - end
199 - end
200 - end
201 - end
202 - end
203 - end
204 - end
205 - end
206 - end
207 - end
208 - end
209 - end
210 - end
211 - end
212 - end
213 - end
214 - end
215 - end
216 - end
217 - end
218 - end
219 - end
220 - end
221 - end
222 - end
223 - end
224 - end
225 - end
226 - end
227 - end
228 - end
229 - end
230 - end
231 - end
232 - end
233 - end
234 - end
235 - end
236 - end
237 - end
238 - end
239 - end
240 - end
241 - end
242 - end
243 - end
244 - end
245 - end
246 - end
247 - end
248 - end
249 - end
250 - end
251 - end
252 - end
253 - end
254 - end
255 - end
256 - end
257 - end
258 - end
259 - end
260 - end
261 - end
262 - end
263 - end
264 - end
265 - end
266 - end
267 - end
268 - end
269 - end
270 - end
271 - end
272 - end
273 - end
274 - end
275 - end
276 - end
277 - end
278 - end
279 - end
280 - end
281 - end
282 - end
283 - end
284 - end
285 - end
286 - end
287 - end
288 - end
289 - end
290 - end
291 - end
292 - end
293 - end
294 - end
295 - end
296 - end
297 - end
298 - end
299 - end
300 - end
301 - end
302 - end
303 - end
304 - end
305 - end
306 - end
307 - end
308 - end
309 - end
310 - end
311 - end
312 - end
313 - end
314 - end
315 - end
316 - end
317 - end
318 - end
319 - end
320 - end
321 - end
322 - end
323 - end
324 - end
325 - end
326 - end
327 - end
328 - end
329 - end
330 - end
331 - end
332 - end
333 - end
334 - end
335 - end
336 - end
337 - end
338 - end
339 - end
340 - end
341 - end
342 - end
343 - end
344 - end
345 - end
346 - end
347 - end
348 - end
349 - end
350 - end
351 - end
352 - end
353 - end
354 - end
355 - end
356 - end
357 - end
358 - end
359 - end
360 - end
361 - end
362 - end
363 - end
364 - end
365 - end
366 - end
367 - end
368 - end
369 - end
370 - end
371 - end
372 - end
373 - end
374 - end
375 - end
376 - end
377 - end
378 - end
379 - end
380 - end
381 - end
382 - end
383 - end
384 - end
385 - end
386 - end
387 - end
388 - end
389 - end
390 - end
391 - end
392 - end
393 - end
394 - end
395 - end
396 - end
397 - end
398 - end
399 - end
400 - end
401 - end
402 - end
403 - end
404 - end
405 - end
406 - end
407 - end
408 - end
409 - end
410 - end
411 - end
412 - end
413 - end
414 - end
415 - end
416 - end
417 - end
418 - end
419 - end
420 - end
421 - end
422 - end
423 - end
424 - end
425 - end
426 - end
427 - end
428 - end
429 - end
430 - end
431 - end
432 - end
433 - end
434 - end
435 - end
436 - end
437 - end
438 - end
439 - end
440 - end
441 - end
442 - end
443 - end
444 - end
445 - end
446 - end
447 - end
448 - end
449 - end
450 - end
451 - end
452 - end
453 - end
454 - end
455 - end
456 - end
457 - end
458 - end
459 - end
460 - end
461 - end
462 - end
463 - end
464 - end
465 - end
466 - end
467 - end
468 - end
469 - end
470 - end
471 - end
472 - end
473 - end
474 - end
475 - end
476 - end
477 - end
478 - end
479 - end
480 - end
481 - end
482 - end
483 - end
484 - end
485 - end
486 - end
487 - end
488 - end
489 - end
490 - end
491 - end
492 - end
493 - end
494 - end
495 - end
496 - end
497 - end
498 - end
499 - end
500 - end
501 - end
502 - end
503 - end
504 - end
505 - end
506 - end
507 - end
508 - end
509 - end
510 - end
511 - end
512 - end
513 - end
514 - end
515 - end
516 - end
517 - end
518 - end
519 - end
520 - end
521 - end
522 - end
523 - end
524 - end
525 - end
526 - end
527 - end
528 - end
529 - end
530 - end
531 - end
532 - end
533 - end
534 - end
535 - end
536 - end
537 - end
538 - end
539 - end
540 - end
541 - end
542 - end
543 - end
544 - end
545 - end
546 - end
547 - end
548 - end
549 - end
550 - end
551 - end
552 - end
553 - end
554 - end
555 - end
556 - end
557 - end
558 - end
559 - end
560 - end
561 - end
562 - end
563 - end
564 - end
565 - end
566 - end
567 - end
568 - end
569 - end
570 - end
571 - end
572 - end
573 - end
574 - end
575 - end
576 - end
577 - end
578 - end
579 - end
580 - end
581 - end
582 - end
583 - end
584 - end
585 - end
586 - end
587 - end
588 - end
589 - end
590 - end
591 - end
592 - end
593 - end
594 - end
595 - end
596 - end
597 - end
598 - end
599 - end
600 - end
601 - end
602 - end
603 - end
604 - end
605 - end
606 - end
607 - end
608 - end
609 - end
610 - end
611 - end
612 - end
613 - end
614 - end
615 - end
616 - end
617 - end
618 - end
619 - end
620 - end
621 - end
622 - end
623 - end
624 - end
625 - end
626 - end
627 - end
628 - end
629 - end
630 - end
631 - end
632 - end
633 - end
634 - end
635 - end
636 - end
637 - end
638 - end
639 - end
640 - end
641 - end
642 - end
643 - end
644 - end
645 - end
646 - end
647 - end
648 - end
649 - end
650 - end
651 - end
652 - end
653 - end
654 - end
655 - end
656 - end
657 - end
658 - end
659 - end
660 - end
661 - end
662 - end
663 - end
664 - end
665 - end
666 - end
667 - end
668 - end
669 - end
670 - end
671 - end
672 - end
673 - end
674 - end
675 - end
676 - end
677 - end
678 - end
679 - end
680 - end
681 - end
682 - end
683 - end
684 - end
685 - end
686 - end
687 - end
688 - end
689 - end
690 - end
691 - end
692 - end
693 - end
694 - end
695 - end
696 - end
697 - end
698 - end
699 - end
700 - end
701 - end
702 - end
703 - end
704 - end
705 - end
706 - end
707 - end
708 - end
709 - end
710 - end
711 - end
712 - end
713 - end
714 - end
715 - end
716 - end
717 - end
718 - end
719 - end
720 - end
721 - end
722 - end
723 - end
724 - end
725 - end
726 - end
727 - end
728 - end
729 - end
730 - end
731 - end
732 - end
733 - end
734 - end
735 - end
736 - end
737 - end
738 - end
739 - end
740 - end
741 - end
742 - end
743 - end
744 - end
745 - end
746 - end
747 - end
748 - end
749 - end
750 - end
751 - end
752 - end
753 - end
754 - end
755 - end
756 - end
757 - end
758 - end
759 - end
760 - end
761 - end
762 - end
763 - end
764 - end
765 - end
766 - end
767 - end
768 - end
769 - end
770 - end
771 - end
772 - end
773 - end
774 - end
775 - end
776 - end
777 - end
778 - end
779 - end
780 - end
781 - end
782 - end
783 - end
784 - end
785 - end
786 - end
787 - end
788 - end
789 - end
790 - end
791 - end
792 - end
793 - end
794 - end
795 - end
796 - end
797 - end
798 - end
799 - end
800 - end
801 - end
802 - end
803 - end
804 - end
805 - end
806 - end
807 - end
808 - end
809 - end
810 - end
811 - end
812 - end
813 - end
814 - end
815 - end
816 - end
817 - end
818 - end
819 - end
820 - end
821 - end
822 - end
823 - end
824 - end
825 - end
826 - end
827 - end
828 - end
829 - end
830 - end
831 - end
832 - end
833 - end
834 - end
835 - end
836 - end
837 - end
838 - end
839 - end
840 - end
841 - end
842 - end
843 - end
844 - end
845 - end
846 - end
847 - end
848 - end
849 - end
850 - end
851 - end
852 - end
853 - end
854 - end
855 - end
856 - end
857 - end
858 - end
859 - end
860 - end
861 - end
862 - end
863 - end
864 - end
865 - end
866 - end
867 - end
868 - end
869 - end
870 - end
871 - end
872 - end
873 - end
874 - end
875 - end
876 - end
877 - end
878 - end
879 - end
880 - end
881 - end
882 - end
883 - end
884 - end
885 - end
886 - end
887 - end
888 - end
889 - end
890 - end
891 - end
892 - end
893 - end
894 - end
895 - end
896 - end
897 - end
898 - end
899 - end
900 - end
901 - end
902 - end
903 - end
904 - end
905 - end
906 - end
907 - end
908 - end
909 - end
910 - end
911 - end
912 - end
913 - end
914 - end
915 - end
916 - end
917 - end
918 - end
919 - end
920 - end
921 - end
922 - end
923 - end
924 - end
925 - end
926 - end
927 - end
928 - end
929 - end
930 - end
931 - end
932 - end
933 - end
934 - end
935 - end
936 - end
937 - end
938 - end
939 - end
940 - end
941 - end
942 - end
943 - end
944 - end
945 - end
946 - end
947 - end
948 - end
949 - end
950 - end
951 - end
952 - end
953 - end
954 - end
955 - end
956 - end
957 - end
958 - end
959 - end
960 - end
961 - end
962 - end
963 - end
964 - end
965 - end
966 - end
967 - end
968 - end
969 - end
970 - end
971 - end
972 - end
973 - end
974 - end
975 - end
976 - end
977 - end
978 - end
979 - end
980 - end
981 - end
982 - end
983 - end
984 - end
985 - end
986 - end
987 - end
988 - end
989 - end
990 - end
991 - end
992 - end
993 - end
994 - end
995 - end
996 - end
997 - end
998 - end
999 - end
1000 - end
1001 - end
1002 - end
1003 - end
1004 - end
1005 - end
1006 - end
1007 - end
1008 - end
1009 - end
1010 - end
1011 - end
1012 - end
1013 - end
1014 - end
1015 - end
1016 - end
1017 - end
1018 - end
1019 - end
1020 - end
1021 - end
1022 - end
1023 - end
1024 - end
1025 - end
1026 - end
1027 - end
1028 - end
1029 - end
1030 - end
1031 - end
1032 - end
1033 - end
1034 - end
1035 - end
1036 - end
1037 - end
1038 - end
1039 - end
1040 - end
1041 - end
1042 - end
1043 - end
1044 - end
1045 - end
1046 - end
1047 - end
1048 - end
1049 - end
1050 - end
1051 - end
1052 - end
1053 - end
1054 - end
1055 - end
1056 - end
1057 - end
1058 - end
1059 - end
1060 - end
1061 - end
1062 - end
1063 - end
1064 - end
1065 - end
1066 - end
1067 - end
1068 - end
1069 - end
1070 - end
1071 - end
1072 - end
1073 - end
1074 - end
1075 - end
1076 - end
1077 - end
1078 - end
1079 - end
1080 - end
1081 - end
1082 - end
1083 - end
1084 - end
1085 - end
1086 - end
1087 - end
1088 - end
1089 - end
1090 - end
1091 - end
1092 - end
1093 - end
1094 - end
1095 - end
1096 - end
1097 - end
1098 - end
1099 - end
1100 - end
1101 - end
1102 - end
1103 - end
1104 - end
1105 - end
1106 - end
1107 - end
1108 - end
1109 - end
1110 - end
1111 - end
1112 - end
1113 - end
1114 - end
1115 - end
1116 - end
1117 - end
1118 - end
1119 - end
1120 - end
1121 - end
1122 - end
1123 - end
1124 - end
1125 - end
1126 - end
1127 - end
1128 - end
1129 - end
1130 - end
1131 - end
1132 - end
1133 - end
1134 - end
1135 - end
1136 - end
1137 - end
1138 - end
1139 - end
1140 - end
1141 - end
1142 - end
1143 - end
1144 - end
1145 - end
1146 - end
1147 - end
1148 - end
1149 - end
1150 - end
1151 - end
1152 - end
1153 - end
1154 - end
1155 - end
1156 - end
1157 - end
1158 - end
1159 - end
1160 - end
1161 - end
1162 - end
1163 - end
1164 - end
1165 - end
1166 - end
1167 - end
1168 - end
1169 - end
1170 - end
1171 - end
1172 - end
1173 - end
1174 - end
1175 - end
1176 - end
1177 - end
1178 - end
1179 - end
1180 - end
1181 - end
1182 - end
1183 - end
1184 - end
1185 - end
1186 - end
1187 - end
1188 - end
1189 - end
1190 - end
1191 - end
1192 - end
1193 - end
1194 - end
1195 - end
1196 - end
1197 - end
1198 - end
1199 - end
1200 - end
1201 - end
1202 - end
1203 - end
1204 - end
1205 - end
1206 - end
1207 - end
1208 - end
1209 - end
1210 - end
1211 - end
1212 - end
1213 - end
1214 - end
1215 - end
1216 - end
1217 - end
1218 - end
1219 - end
1220 - end
1221 - end
1222 - end
1223 - end
1224 - end
1225 - end
1226 - end
1227 - end
1228 - end
1229 - end
1230 - end
1231 - end
1232 - end
1233 - end
1234 - end
1235 - end
1236 - end
1237 - end
1238 - end
1239 - end
1240 - end
1241 - end
1242 - end
1243 - end
1244 - end
1245 - end
1246 - end
1247 - end
1248 - end
1249 - end
1250 - end
1251 - end
1252 - end
1253 - end
1254 - end
1255 - end
1256 - end
1257 - end
1258 - end
1259 - end
1260 - end
1261 - end
1262 - end
1263 - end
1264 - end
1265 - end
1266 - end
1267 - end
1268 - end
1269 - end
1270 - end
1271 - end
1272 - end
1273 - end
1274 - end
1275 - end
1276 - end
1277 - end
1278 - end
1279 - end
1280 - end
1281 - end
1282 - end
1283 - end
1284 - end
1285 - end
1286 - end
1287 - end
1288 - end
1289 - end
1290 - end
1291 - end
1292 - end
1293 - end
1294 - end
1295 - end
1296 - end
1297 - end
1298 - end
1299 - end
1300 - end
1301 - end
1302 - end
1303 - end
1304 - end
1305 - end
1306 - end
1307 - end
1308 - end
1309 - end
1310 - end
1311 - end
1312 - end
1313 - end
1314 - end
1315 - end
1316 - end
1317 - end
1318 - end
1319 - end
1320 - end
1321 - end
1322 - end
1323 - end
1324 - end
1325 - end
1326 - end
1327 - end
1328 - end
1329 - end
1330 - end
1331 - end
1332 - end
1333 - end
1334 - end
1335 - end
1336 - end
1337 - end
1338 - end
1339 - end
1340 - end
1341 - end
1342 - end
1343 - end
1344 - end
1345 - end
1346 - end
1347 - end
1348 - end
1349 - end
1350 - end
1351 - end
1352 - end
1353 - end
1354 - end
1355 - end
1356 - end
1357 - end
1358 - end
1359 - end
1360 - end
1361 - end
1362 - end
1363 - end
1364 - end
1365 - end
1366 - end
1367 - end
1368 - end
1369 - end
1370 - end
1371 - end
1372 - end
1373 - end
1374 - end
1375 - end
1376 - end
1377 - end
1378 - end
1379 - end
1380 - end
1381 - end
1382 - end
1383 - end
1384 - end
1385 - end
1386 - end
1387 - end
1388 - end
1389 - end
1390 - end
1391 - end
1392 - end
1393 - end
1394 - end
1395 - end
1396 - end
1397 - end
1398 - end
1399 - end
1400 - end
1401 - end
1402 - end
1403 - end
1404 - end
1405 - end
1406 - end
1407 - end
1408 - end
1409 - end
1410 - end
1411 - end
1412 - end
1413 - end
1414 - end
1415 - end
1416 - end
1417 - end
1418 - end
1419 - end
1420 - end
1421 - end
1422 - end
1423 - end
1424 - end
1425 - end
1426 - end
1427 - end
1428 - end
1429 - end
1430 - end
1431 - end
1432 - end
1433 - end
1434 - end
1435 - end
1436 - end
1437 - end
1438 - end
1439 - end
1440 - end
1441 - end
1442 - end
1443 - end
1444 - end
1445 - end
1446 - end
1447 - end
1448 - end
1449 - end
1450 - end
1451 - end
1452 - end
1453 - end
1454 - end
1455 - end
1456 - end
1457 - end
1458 - end
1459 - end
1460 - end
1461 - end
1462 - end
1463 - end
1464 - end
1465 - end
1466 - end
1467 - end
1468 - end
1469 - end
1470 - end
1471 - end
1472 - end
1473 - end
1474 - end
1475 - end
1476 - end
1477 - end
1478 - end
1479 - end
1480 - end
1481 - end
1482 - end
1483 - end
1484 - end
1485 - end
1486 - end
1487 - end
1488 - end
1489 - end
1490 - end
1491 - end
1492 - end
1493 - end
1494 - end
1495 - end
1496 - end
1497 - end
1498 - end
1499 - end
1500 - end
1501 - end
1502 - end
1503 - end
1504 - end
1505 - end
1506 - end
1507 - end
1508 - end
1509 - end
1510 - end
1511 - end
1512 - end
1513 - end
1514 - end
1515 - end
1516 - end
1517 - end
1518 - end
1519 - end
1520 - end
1521 - end
1522 - end
1523 - end
1524 - end
1525 - end
1526 - end
1527 - end
1528 - end
1529 - end
1530 - end
1531 - end
1532 - end
1533 - end
1534 - end
1535 - end
1536 - end
1537 - end
1538 - end
1539 - end
1540 - end
1541 - end
1542 - end
1543 - end
1544 - end
1545 - end
1546 - end
1547 - end
1548 - end
1549 - end
1550 - end
1551 - end
1552 - end
1553 - end
1554 - end
1555 - end
1556 - end
1557 - end
1558 - end
1559 - end
1560 - end
1561 - end
1562 - end
1563 - end
1564 - end
1565 - end
1566 - end
1567 - end
1568 - end
1569 - end
1570 - end
1571 - end
1572 - end
1573 - end
1574 - end
1575 - end
1576 - end
1577 - end
1578 - end
1579 - end
1580 - end
1581 - end
1582 - end
1583 - end
1584 - end
1585 - end
1586 - end
1587 - end
1588 - end
1589 - end
1590 - end
1591 - end
1592 - end
1593 - end
1594 - end
1595 - end
1596 - end
1597 - end
1598 - end
1599 - end
1600 - end
1601 - end
1602 - end
1603 - end
1604 - end
1605 - end
1606 - end
1607 - end
1608 - end
1609 - end
1610 - end
1611 - end
1612 - end
1613 - end
1614 - end
1615 - end
1616 - end
1617 - end
1618 - end
1619 - end
1620 - end
1621 - end
1622 - end
1623 - end
1624 - end
1625 - end
1626 - end
1627 - end
1628 - end
1629 - end
1630 - end
1631 - end
1632 - end
1633 - end
1634 - end
1635 - end
1636 - end
1637 - end
1638 - end
1639 - end
1640 - end
1641 - end
1642 - end
1643 - end
1644 - end
1645 - end
1646 - end
1647 - end
1648 - end
1649 - end
1650 - end
1651 - end
1652 - end
1653 - end
1654 - end
1655 - end
1656 - end
1657 - end
1658 - end
1659 - end
1660 - end
1661 - end
1662 - end
1663 - end
1664 - end
1665 - end
1666 - end
1667 - end
1668 - end
1669 - end
1670 - end
1671 - end
1672 - end
1673 - end
1674 - end
1675 - end
1676 - end
1677 - end
1678 - end
1679 - end
1680 - end
1681 - end
1682 - end
1683 - end
1684 - end
1685 - end
1686 - end
1687 - end
1688 - end
1689 - end
1690 - end
1691 - end
1692 - end
1693 - end
1694 - end
1695 - end
1696 - end
1697 - end
1698 - end
1699 - end
1700 - end
1701 - end
1702 - end
1703 - end
1704 - end
1705 - end
1706 - end
1707 - end
1708 - end
1709 - end
1710 - end
1711 - end
1712 - end
1713 - end
1714 - end

```



NPTEL

```
rapid_decoder.m
131 % if no input == 1 step 1 and go to left child
132 % temp = P*temp
133 % ln = Ldepth*2, temp*node(ltemp*(node+1))r binomial beliefs
134 % a = Ldepth*2/r b = Ldepth*2/r*lendr logit beliefs into 2
135 % node = node*2^r depth + 1 r next node left child
136 % Temp = temp / 2r binomial belief length for left child
137 % Ldepth*2/r*temp*node(ltemp*(node+1)) = (x,y) balloon and stc
138 % ninput = 1r
139 %
140 % else
141 % if ninput == 1 step R and go to right child
142 % temp = 2^r*depth
143 % ln = Ldepth*2, temp*node(ltemp*(node+1))r binomial beliefs
144 % a = Ldepth*2/r b = Ldepth*2/r*lendr logit beliefs into 2
145 % node = node*2^r depth + 1r next child
146 % Temp = temp / 2r binomial belief length for right child
147 % Ldepth*2/r*temp*node(ltemp*(node+1)) = g(x,y)wappa
148 % ninput = 2r
149 %
150 % else step 3 and go to parent
151 % temp = 2^r*depth
152 % ln = 2^r*depth*node + 2^r*node*cldepth*depth + 1r
153 % cldepth = 1r
154 % ninput = 1r
155 % ninput = 1r
156 % ninput = 1r
157 % ninput = 1r
158 % ninput = 1r
159 % ninput = 1r
160 %
161 end
162 end
```

PROF. ANDREW THANGARAJ

DT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 2



NPTEL

```
rapid_decoder.m
85 % ninput=1, node(1)=0
86 %
87 % if ltemp, node(1) == 0
88 % ninput(1, node(1)) = 0
89 %
90 % else
91 % ninput(1, node(1)) = 1
92 %
93 % end
94 %
95 % if node == (N-1)
96 % node = 2^r step 1 and go to left child
97 % temp = P*temp
98 % ln = Ldepth*2, temp*node(ltemp*(node+1))r binomial beliefs
99 % a = Ldepth*2/r b = Ldepth*2/r*lendr logit beliefs into 2
100 % node = node*2^r depth + 1r next node left child
101 % Temp = temp / 2r binomial belief length for left child
102 % Ldepth*2/r*temp*node(ltemp*(node+1)) = (x,y) balloon and stc
103 % ninput = 1r
104 %
105 % else
106 % if ninput == 1 step R and go to right child
107 % temp = 2^r*depth
108 % ln = Ldepth*2, temp*node(ltemp*(node+1))r binomial beliefs
109 % a = Ldepth*2/r b = Ldepth*2/r*lendr logit beliefs into 2
110 % node = node*2^r depth + 1r next child
111 % Temp = temp / 2r binomial belief length for right child
112 % Ldepth*2/r*temp*node(ltemp*(node+1)) = g(x,y)wappa
113 % ninput = 2r
114 %
115 end
116 end
```

PROF. ANDREW THANGARAJ

DT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 2



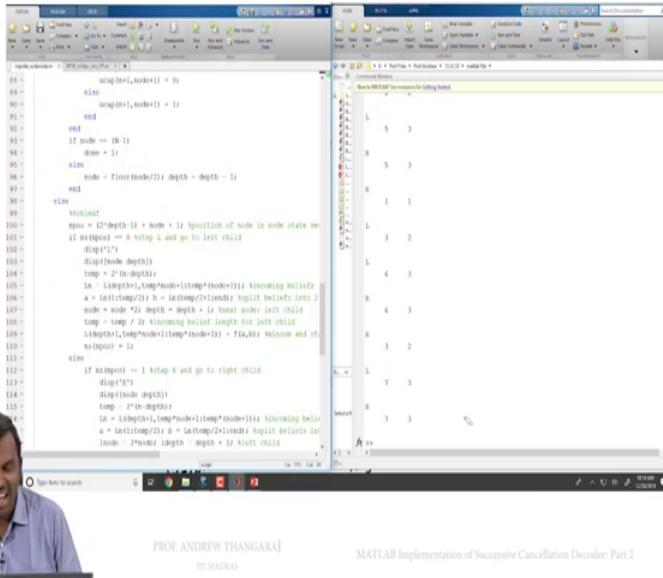
NPTEL

```
rapid_decoder.m
74 % l = R(x,y) = 2^r*(a+b*x)^{1/2}*(y+c*x)^{1/2} min(a,x,y)0.000000
75 % q = R(x,y)*x^2*(1-2*x)^2*4*x^4 by formula
76 %
77 % Ldepth = 0.0. Node of root
78 %
79 % node = 0 depth at root
80 % node > 0 known as r known or not
81 % while r > 0 iteration till all bits are decoded
82 % fixed or not
83 % if depth == n
84 % if any(r>node(1)) r is node from
85 % ninput(r, node(1)) = 0r
86 %
87 % if Ldepth*2/r*temp*node(ltemp*(node+1)) > 0
88 % ninput(r, node(1)) = 0r
89 % else
90 % ninput(r, node(1)) = 1r
91 %
92 % end
93 % if node == (N-1)
94 % node = 1r
95 %
96 % else
97 % node = node*2^r depth + 1r
98 % ninput(r, node(1)) = 0r
99 %
100 % ninput(r, node(1)) = 1r
101 %
102 % if no input == 1 step 1 and go to left child
103 % disp('1')
104 % disp(node depth)
```

PROF. ANDREW THANGARAJ

DT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 2



MATLAB Implementation of Successive Cancellation Decoder: Part 2

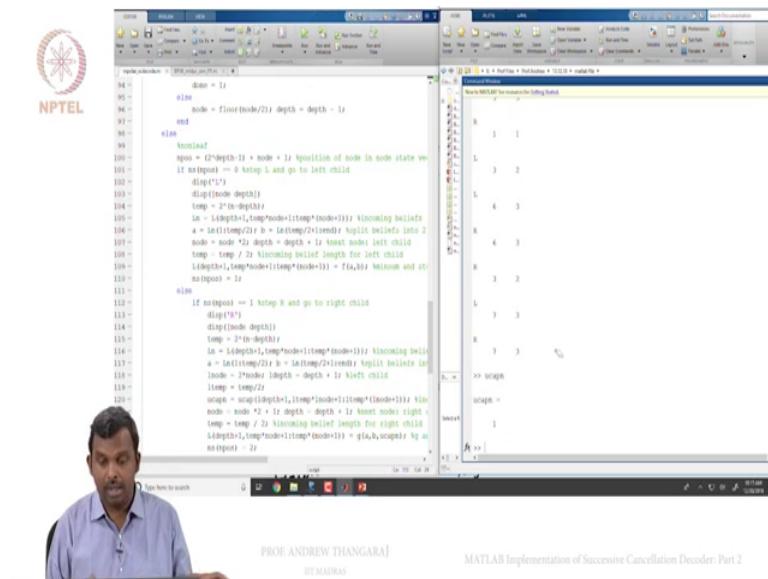
So let us take up the A and B okay A is 2 values now okay, so 0.5684, 0.2086 then B is the remaining two values 0.1285 and 0.4985 okay, so node okay for some reason I calculate it, I have gone to the next node okay alright, so this is fine, so this is good, so then you have U cap N which will be actually two 0s and then when I computes it, it will go off and then the next state goes to this.

So I have not gone to the next node, that is the next node and going to, so now this node is 1 and 2 okay, so this is okay, so this is looking correctly, so you have the Ls calculated okay, so you can go through and look at this and see how it proceeds then this will not be true, this will go left further again yes and then it will come back make a decision then it will go right again, just to check and then it will go, make a decision it is doing a lot of things.

So it seems to be working, there is no major bug here at least I am not checked it fully but at least is working, so I am going to clear all the breakpoints and just to check what is happening, so I am going to print some information out just to check that things are happening correctly or wrongly, so any time I enter my left I am going to print, print does work, so I am going to just display node and depth okay, so just to make sure I see something every time I go left and so I should say I went left right.

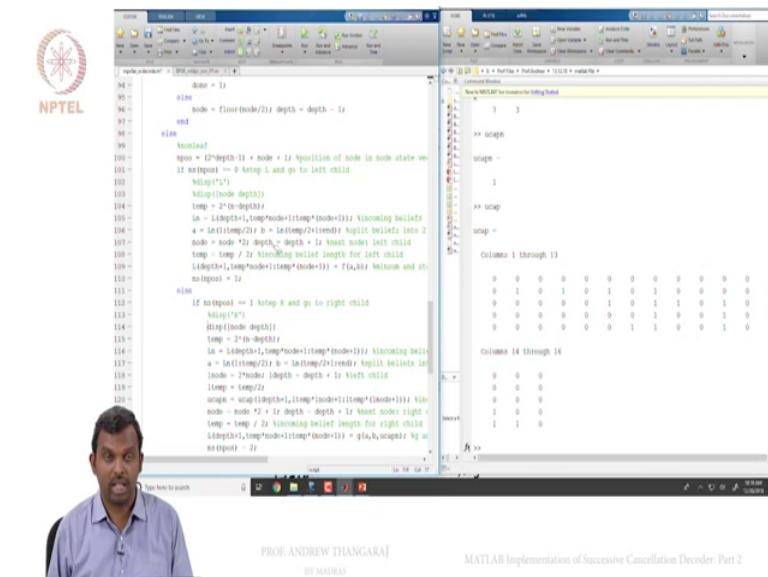
So I display left also okay, then I say when I go right I will display right just to see that whole thing is proceeding correctly or wrong and then depth and, node and depth okay just to see how the whole thing is going, than it should finish, so let me just run it once to see what it does okay, so it finished give me some answer but no idea whether the answer is correct or wrong but at least it ran.

(Refer Slide Time: 31:36)



So let us see whether the sequence is correct or wrong okay, so that is the first thing one can check, L node 0 okay, so it went left to depth 1, went left to depth 2, went left to depth 3 then it went right okay to depth 3, right from depth 3 and then right from depth 2 left from, so it looks okay and it is given me some final answer, so you can check on $U \cap N$.

(Refer Slide Time: 32:07)



So if you look at the U cap, this is the overall U cap but remember the actual message is in the last row okay I did not pull it out in the code but the actual message is in the last row, I was just checking here to ensure that everything is happening correctly, it looks to be working fine so the message is in the last row okay.

So let us maybe not print is things, this is just some debugging to check everything is working correctly, it seems to be working correctly, so I am done with the decoder as I now I can assign my M cap right, so my actual, so I have to pull out the messages from the frozen positions or I could just compare the U cap of the last one with the U right, so U is my actual code word or code word itself, so that is something that one can do, let me save this okay.

(Refer Slide Time: 33:08)

The screenshot shows a MATLAB interface. On the left is a code editor with MATLAB code for a Successive Cancellation Decoder. On the right is a command window showing the results of running the code. The code includes comments explaining the steps of the decoding process, such as calculating log-likelihood ratios (LLRs), selecting the best path, and updating belief states. The command window displays binary matrices representing the state of nodes at different stages of the tree, with labels for columns 1 through 14.

PROF. ANDREW THANGARA
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 2

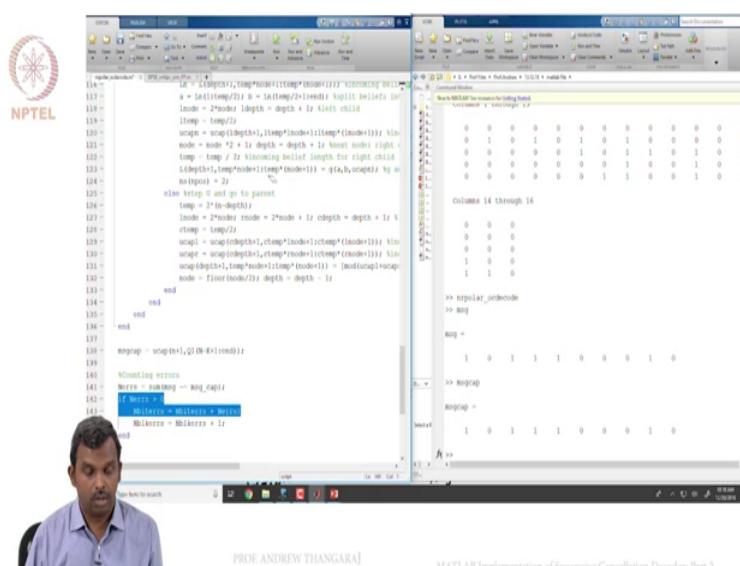
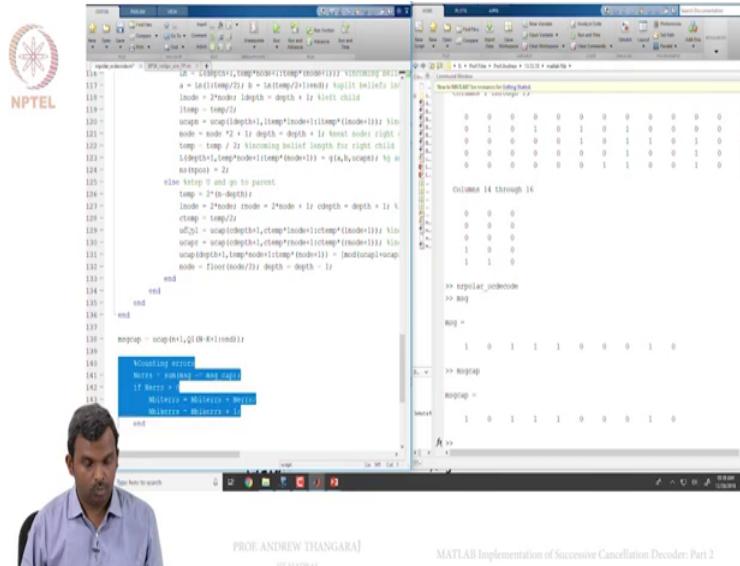
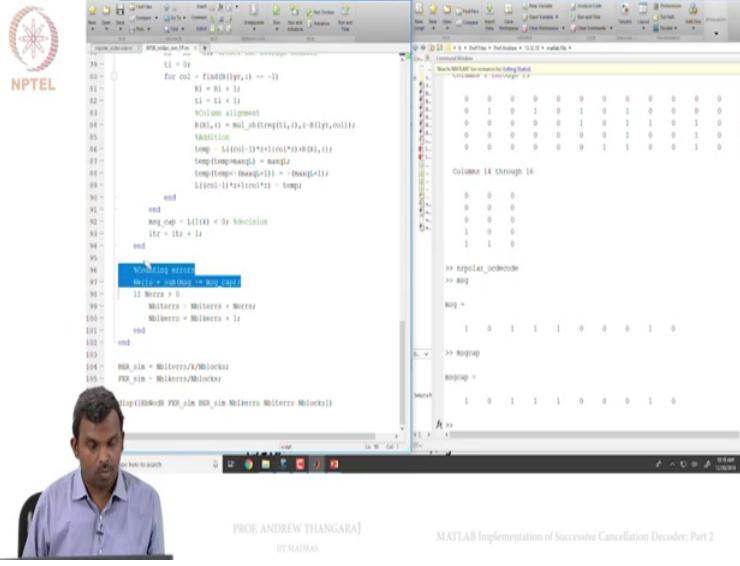
So you can see if these two are equals, so the message went into view of this right, so this is where the message went in to okay, so if I pull out this I should get the message cap back okay, so let us that U cap of N plus 1, the same Q1, there was the Q1 of N minus K plus 1 colon N okay, so this is the position where the thing got inserted, so the same thing in the last row should be my message cap okay.

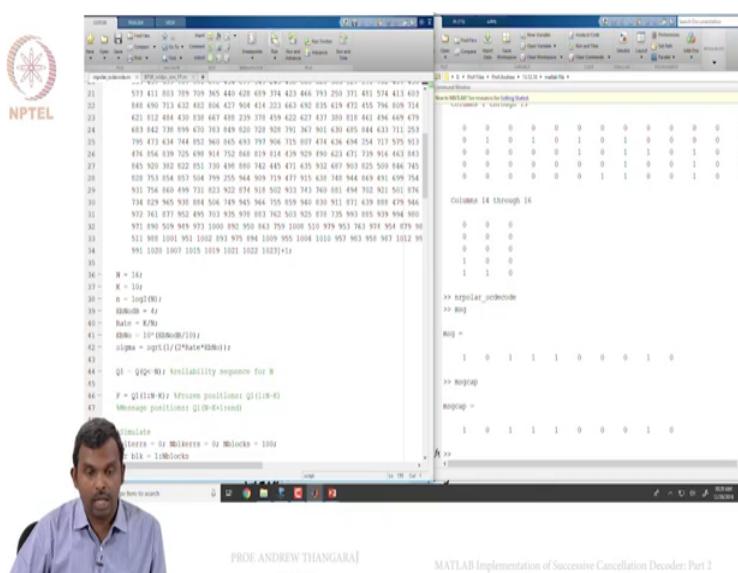
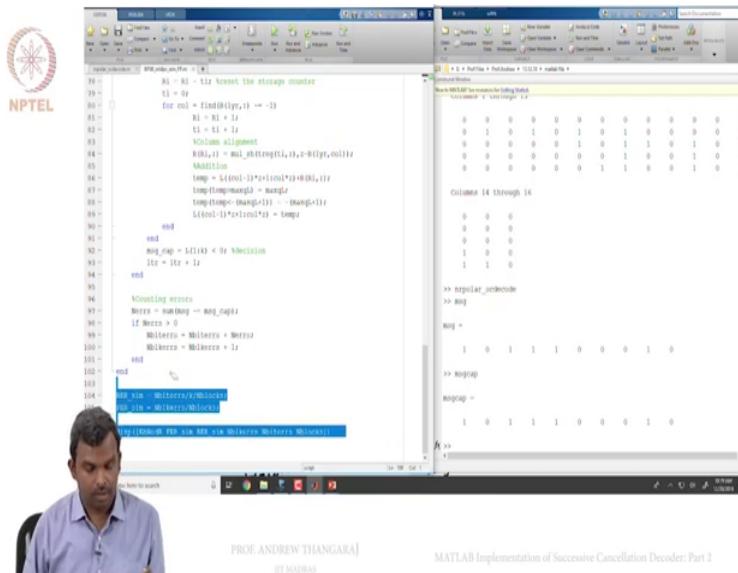
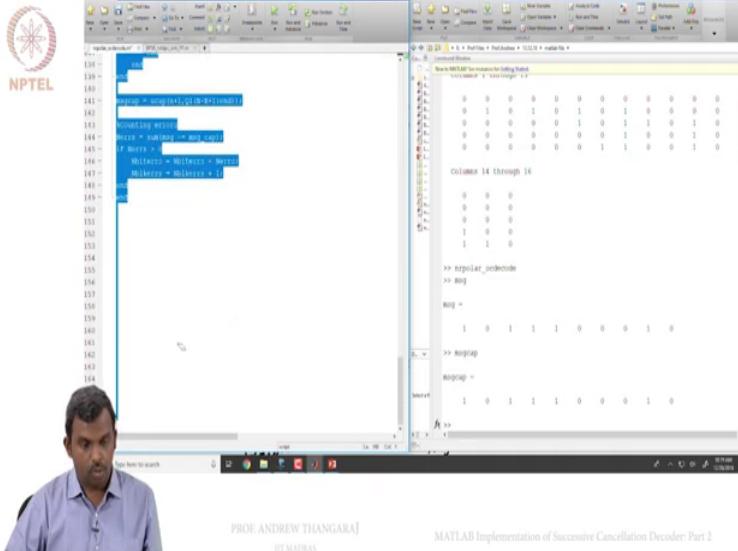
(Refer Slide Time: 33:58)

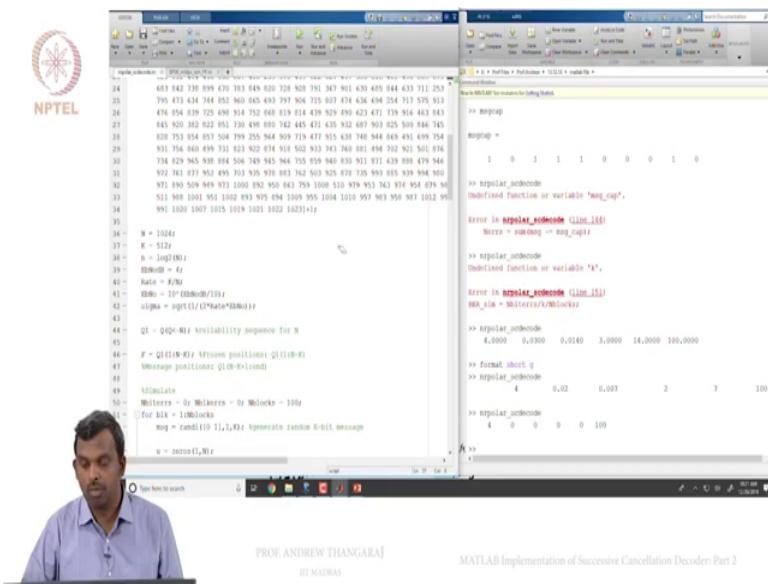
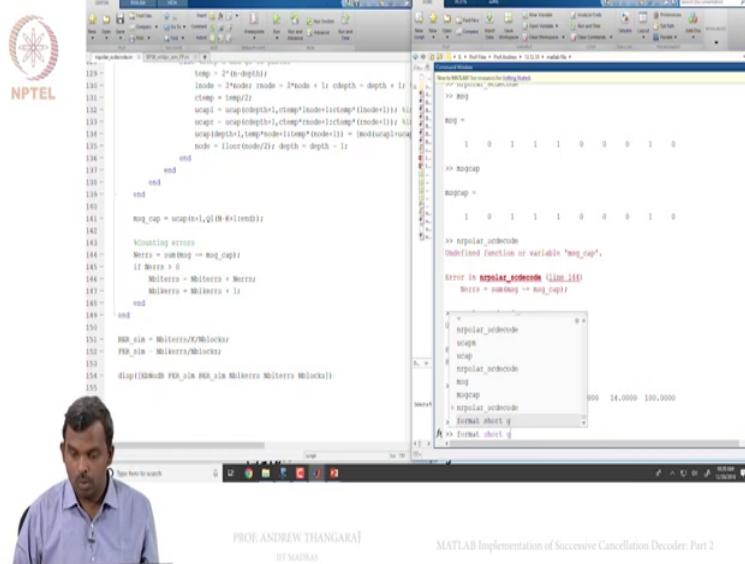
This screenshot is similar to the one above, showing the same MATLAB code and command window. The command window now shows the final output, which includes the message cap (B01) and the final message sequence (B001101110001). The professor is visible in the foreground, gesturing with his hand near his ear, possibly indicating he is listening or thinking.

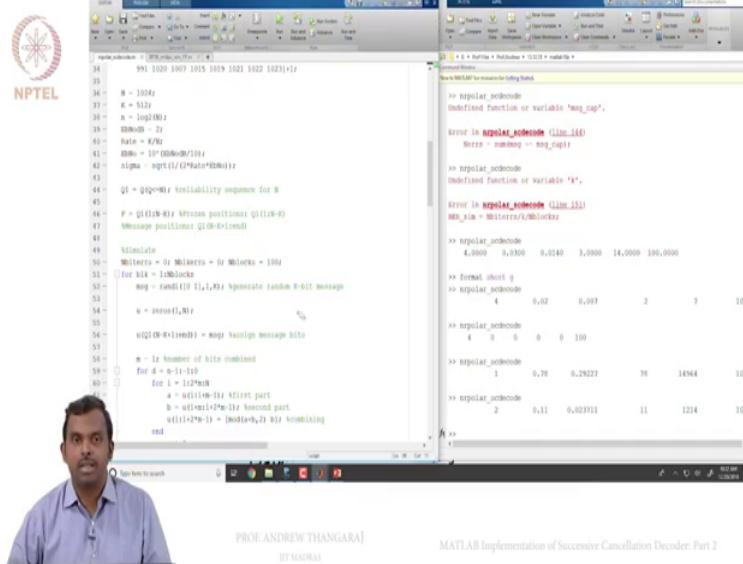
PROF. ANDREW THANGARA
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 2









So let us run this and then see if we're at this highest NR if it is working or not, you say you had a message and you had a message cap and they are the same very nice is not it, so it is good to see that the message in the message cap agree maybe there were no errors in the channels but still our decoder is working and it is agreeing, so you can again to the same thing as before our way in which count the errors, you remember I do the error counting okay, so you can do that okay, so I am not putting it in a loop.

So this part is not so relevant but anyway, so I am not putting it in a loop for calculating the message and sending it out, so maybe that part we can add, so how do you add that, so you put this here and blocks, where do I put this, I needed below this simulate and then I need a for loop, for loop needs an N and you know the whole thing can be intended, intended a little bit so that it goes off that key okay, so I think I have done most of what I wanted to do.

So I put blockers 0 and then I am simulating for every block and every block I generate a message, I had code word and I transmitted, I assigned these things to 0, I think I am doing all of that correctly than simulating it, decoding it, getting a message cap and checking it up okay and then we can now print, the B are same and the, and this, there you go.

So this will even simulate 100 blocks for the 1610 code at 4 DB SNR so let us do that, there you go okay, did I not put underscore okay sorry, there you go, it got to change the capital K okay, so now we are ready to simulate and you get some answers okay, so there were 3 block errors out of 100, format short G, opps what did I do.

So one can decode, so you get 2 errors out of 100, 3 errors out of 100, so this is basically working, so maybe we want to be very more experimental, so let us try the largest block

length that the 5G standard allows, 1024 and 512, so this is going to be big and let us see what it does, no errors at 4 DB is huge SNR, so let us make it maybe 1 DB, maybe will see some errors out of 100, 78 were an errors, so maybe 1 DB is not too great, maybe 2 DB, 11 were an errors so on.

So the basic decoder I believe is working, this is the successive cancellation decoder for polar codes and you saw it is reasonably easy to code, it is a short piece of code that gives you good answers okay of course one can make it more efficient, one can make it more interesting, one can make modifications we can keep on working on it and what we will do the next week is to look at list decoding for polar codes, so list decoding is quite important that is the decoder which actually works in practice, it gives you better gains than this, will see that in the next class, in the next lecture and as far as this lecture is concern we are done. Thank you very much.