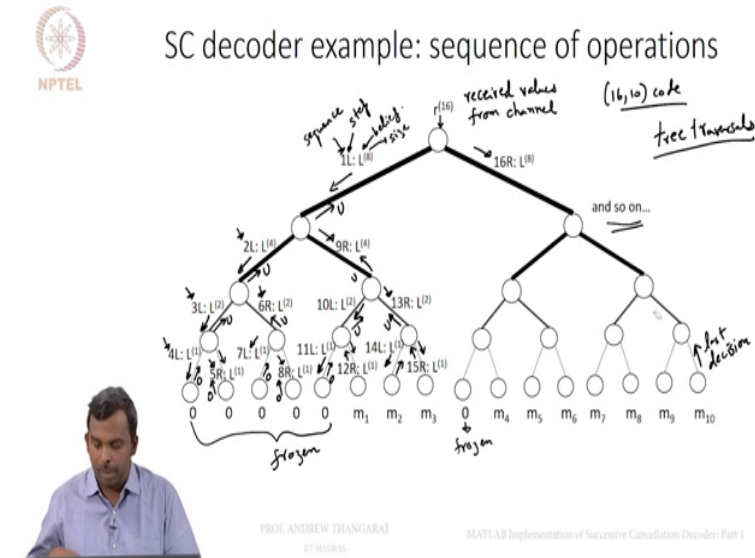


LDPC & Polar codes in 5G Standard
Prof. Andrew Thangaraj
Department of Electrical Engineering
Indian Institute of Technology Madras
MATLAB Implementation of Successive Cancellation Decoder Part 1

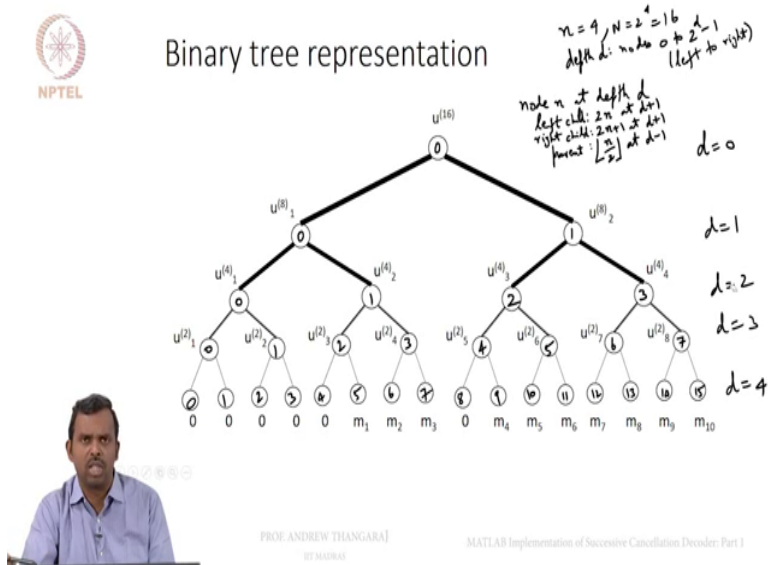
(Refer Slide Time: 0:20)



Okay hello and welcome to this lecture and in this lecture we will write a simple piece of MATLAB code for implementing this successive cancellation decoder, once again I want to emphasise you not writing very efficient code which could be probably directly implemented on hardware or something, we are just writing something that works okay, so that you can play around with it and use it for other sort of things okay.

Okay, so what will it take to implement this decoder? Of course you need a lot of representation right, inside the code how will you represent etc.

(Refer Slide Time: 0:50)



First think you need to represent is the tree itself, so how do you represent a binary tree? So like I said, I will do it the first thing I will use is this notion of depth, so this has depth equals 0, this level and then you have depth equals 1 and depth equals 2, depth equals 3, depth equals 4 okay, so remember this one is n equals 4 and N equals to power 4 which is 16 okay, so that is this example, if you have a more general one the depth will go further and further.

So for instance if it is 1024 the depth will go from D equals 0 all the way up to D equals 10 okay, so depth is one thing, so this tells you what depth you are and then what about the node okay, the node numbering will start with 0 and then go, 0,1,0,1,2,3, and then 0,1,2,3,4,5,6,7 and then 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 okay, so these are the numbers okay, so at any depth D you have nodes 0 to $2^D - 1$ okay, so it is easy to see, so even for D equals 0 this is true right.

So if you have depth D you have node 0 to $2^D - 1$ from left to right, that is one thing, so this is easy enough I have represented each node and then from each node I need to know what its parent is right, and I need to know what its left child is and what is right child is okay, so if you have node N at depth D okay, so what is the left child? What is the right child and what is the parent? That is my question, so you have, so avoid some trivial cases, so will assume it is some sort of an interior node, so that you do not have the issue of leaf or root, root has no parent, leaf has no children okay.

So will assume some interior node, so that node N and depth D , what is the left child? What is the right child and what is the parent? Okay, so you can look at it very very closely, there is



SC decoder example: sequence

Beliefs

$d=0$ node 0 16 beliefs

$d=1$ node 0 8 beliefs node 1 8 beliefs

$d=2$ node 0 node 1 node 2 node 3
4 4 4 4

2D array $L[0:4, 0:15]$



PROF. ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 1



SC decoder example: sequence of operations

Beliefs

$d=0$ node 0 16 beliefs

$d=1$ node 0 8 beliefs node 1 8 beliefs

$d=2$ node 0 node 1 node 2 node 3
4 4 4 4

$[0:4, 0:15]$

at depth d : incoming beliefs
 $L[d,]$



PROF. ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 1



SC decoder example: sequence

Beliefs

$N=2^2$

$d=0$ node 0 16 beliefs node t node $t-d$
inferred beliefs

$d=1$ node 0 8 beliefs node 1 8 beliefs

$d=2$ node 0 node 1 node 2 node 3
4 4 4 4

2D array $L[0:4, 0:15]$

node i at depth d : incoming beliefs
 $L[d, i]$



PROF. ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 1

NPTEL

SC decoder example: sequence

Handwritten notes on whiteboard:

- d=0 16 beliefs
- d=1 8 beliefs
- d=2 4 beliefs
- d=3 2 beliefs
- node 0, node 1, node 2, node 3
- 2D array $L[d, i: 2^i-1]$
- node i at depth d: incoming beliefs $L[d, i: 2^i-1]$

PROF. ANDREW THANGARAJ
@MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 1

And the next thing is the messages okay, so what are the messages that need to be stored, so if you remember in the encoder, we had U at the bottom and then it got pushed into the next step, next step, next step and all that till it got to the root and there was no need to store any intermediate things, it was just going in one direction but the decoder does not go in one direction okay.

Excuse me, the decoder does not go in one direction, it does not go just down okay, it goes down then it comes back up and remember when it comes back up every node still needs to remember what is its incoming message was okay, you cannot forget that because of for instance here if you look at this node okay, I first receives 8 beliefs okay, these are some eight numbers okay, it processes it and sends out 4 beliefs to its left child, so it cannot forget its incoming 8 beliefs, why is that? Because when the left child send back its decision, it uses the incoming 8 beliefs and computes the G function okay.

So you have to do that okay but of course when both the left and right child have send their decisions you can forget about what is their okay, so it is possible, so you can, you do not have to store everything but we will be a bit relaxed about it and we will store all the messages that are flowing in every depth okay, so if you see here, you have depth 0 okay, you have beliefs, there are 16 beliefs okay, maybe I should write it a bit differently.

So what I will do is, I will keep this as a shorter window, not a full screen window can also open up journal here so that I can do something, I can write on journal as well and then you will see something little bit better okay, so let us do this okay, so this is a slide, so what about the received value, so you if you look at, this is the belief okay, so you have to store the

beliefs, so how do you store the beliefs in the decoders, so if you look at depth 0 okay and node 0 have 16 beliefs incoming okay it needs to store all that.

Then if you look at depth 1, node 0 and depth 1 has 8 beliefs and then node 1 has 8 beliefs, these are all incoming, node 1 is a sort of node immediate, eventually it happens but it has 8 beliefs, likewise said D equals 2 you can see node 0, node 1, node 2, node 3, they have 4 beliefs incoming each okay, so basically in every depth you have 16 beliefs okay, in every depth you have 16 beliefs, they are associated with different nodes but the total is 16 okay.

So what we will do is, we will have two dimensional array L of let us say in this case 0: L I use this notation is 0:3 okay and then actually 4 right, so it go all the way up to 4 and then 0:15 okay, this will be the 2 minus D array but remember I am indexing from 0 and Matlab you have to do that minor adjustment to index from 1, so I will just, I will use 0 for now and you will see later on it easy to make that adjustment in Matlab okay.

So I have 2 minus D array which stores all this beliefs okay, so that is important, so once you store this beliefs, you can associate the nodes with them, so for instance node N at depth D okay, its incoming beliefs, these are all incoming beliefs okay are in L of D , okay four node N it will be, so 2^D if you remember is the incoming, so maybe I should say that as well right maybe say that, for a node at depth D the incoming belief I mean, so I have done that here, so it is basically 2^D right.

So node at depth D has, it is not 2^D , it is 2^{N-D} okay, so let me just use this notations 2^{N-D} okay, so I am using too many N s here, so maybe I make this node I okay, so N is 2^{N-D} okay, so if you have depth D node 0 has 16 beliefs right, so node at depth D has 2^{N-D} incoming beliefs okay, so hopefully that is clear to you have and so if you look at a node I at depth D the array entry it will correspond to its L of D , it will be in the D th row and then what will be the columns that will be important.

So it will be 2^{N-D} okay, so times $I: 2^{N-D}$ times I plus $2^{N-D} - 1$ okay, so this will be the 2^{N-D} beliefs associated with the node I okay, so hopefully that is clear to you, so it for instance maybe I can use this example, so if you have L here of this, if you think offer array here so it will have the entry 0,1,2,3 associated with node 0 and then 4,5,6,7 these four associated with node 1 okay and then 8,9,10,11 this four are associated with node 2 and then 12,13,14,15 this four are associated with node 3 okay, so this is D th row of L okay, hopefully that is easy enough to see


So you just group all of them together in one array but you associate different blocks to different nodes, so this is just for ease of storage so that we can refer to this when we want okay, hopefully this is clear, so we will use this array L which will be a two dimensional array, so in general it will be L of 0 colon N , 0 colon $2^N - 1$ okay, so that will be the size of this array and the depth D is associated with 2^{N-D} incoming beliefs and how do you store those two party I mean you store it as one row vector but different chunks of it you associate with different nodes okay.

So this is how we do the beliefs okay, so we have represented the tree we know how to go from one node to the other in either left child, right child or parent, we have figure out how to store the beliefs okay, maybe not the most efficient, I am sure you can find a lot holes in how you can simplify the storage, you do not need to store is much etc, while like I said I am not trying to write the most efficient code here and the third piece of information that we need is some sort of a status for the node okay.


So if you remember a node comes into operation only once in a while, it gets activated and it can be in different states right, the state of a node it may be never mean, maybe it never been activated so far, maybe it has finished step L , it is waiting for step R maybe it has finished step R also it is waiting for step U maybe it has finish step U also, it just done okay, so there are multiple states of the nodes okay.

So for every node we have to store the state okay, so like I said there are three things you have to worry about, one is representing the binary tree being able to traverse we figure that out, next is storing the beliefs all through the tree and we are doing some extra storage, it is being very generous here and then the third part is storing the states of the node, every node will be indifferent state as it goes through the decoder okay, so you have to store the state okay.

(Refer Slide Time: 13:54)



SC decoder example: sequence



node 0 node 1 node 2 node 3


node i at depth d: incoming beliefs $L[d, 2^i : 2^{i+1}-1]$

States of nodes:


- 0: yet to be activated
- 1: finished L
- 2: finished R
- 3: finished U

PROF ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 1



SC decoder example: sequence




State vector

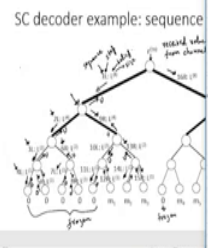
node i at depth d: position $(2^i-1)+i$
 last node: $d=11$ and node $2^{11}-1$
 position: 2^i-1+2^{i-1}
 $= 2(2^{i-1})$
 $= 2^{i-1}$

PROF ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 1



SC decoder example: sequence



States of nodes:

- 0: yet to be activated
- 1: finished L
- 2: finished R
- 3: finished U

State vector

node i at depth d: position $(2^i-1)+i$
 last node: $d=11$ and node $2^{11}-1$
 position: 2^i-1+2^{i-1}
 $= 2(2^{i-1})$
 $= 2^{i-1}$

PROF ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 1

So states, we will use a slightly different notation just to be a bit more efficient than just you know just using some big 2 minus D array, we could use a big 2 minus D array as well but I will be a little bit more efficient the nodes to make things easy okay, so let us first summarise what the states are 0 is yet to be activated okay, 1 is finished L okay, 2 is finished, R3 is finished U okay, this is just my notation there are 4 states to the nodes, it could be 0 not yet activated, so initially all states will be 0 means all nodes will be in state 0 okay and then once you have done L it goes on to state 1 and once you have done R for that node it goes into state 2, once you have done U for that node, unit what is LR and UR, the three different steps that we do on the node goes into state 3 okay.

Once it goes into state 3 it is done okay, so this is how you store the node and then how I going to store the states okay, so how do I store the states, so maybe I do something slightly efficient here instead of being very very inefficient, so let us see so we have, remember so at depth 0 you have only 0 okay, so I will store the states in in one linear array okay, so initially I have depth 0 and then I have, this is depth 0 then I have depth 1, at depth 1 I have only 2 nodes right, 0 and 1, so this 2 are depth 1 and then I have depth 2, I have 0, 1, 2, 3 okay and then I have depth 3, I have 0, 1, 2, 3, 4, 5, 6, 7 and so on right.

So this is what happens and so on, so in general node I at depth D will be at position, the position is, if you look at depth 3 it has 7 before, so it is going to be 2 power D minus 1 okay, so that is 7 plus I okay, so remember the overall count is 0 or 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 and so on okay that I miss anything okay, is one thing I miss anything, so it is goes on like this okay, so again I remember I am indexing from 0, this is one adjustment that you will have to make when we go to MATLAB change that.

So node I at depth D is at position is 2 power D minus 1 plus I okay, so if you think of depth 2 and node 2 it will be 3 plus 2, 5, so you will get the correct numbers, node 7 at depth 3 it will be 7 plus 7 which is 14 okay, so all of that will happen this is, so this is the state vector okay and what will be the total length you can compute, so what is the last node D equals N, the last node will be D equals N right depth of N and node 2 power N minus 1 right, so node 2 power N minus 1 it depth 2 N, so that has position 2 power N minus 1 plus 2 power N minus 1 which is 2 x 2 power N minus 1 which is 2 power N plus 1 minus 2 okay, so the overall length, so it will go from 0 to 2 power N plus 1 minus 2 okay, so the length is this plus 1 which is 2 power N plus 1 minus 1, so that is the total length we need.

So in fact there is the total number of nodes in the binary tree okay, so the leaf has 2^N minus, 2^N minus 2^N by N minus 1 so on till 1, so that will add up to 2^N plus 1 minus 1 okay, so that is the total number of nodes, so will be a little efficient in storing the states, so once again what is the state vector, initially it will store all 0s and ones a particular node has completed a certain operation it will switch into either 1, 2 or 3 okay.

So with this we are ready to start writing the programs, so now it has been known how to store represent binary tree nodes and traverse from one to the other we know that, we know how to store the beliefs and how to access the beliefs corresponding to a particular node, we know how to store the states of the nodes and access the state of a particular node and we can start writing the decoder okay, let us see.

(Refer Slide Time: 19:20)

The image shows a MATLAB environment. On the left, a file explorer window displays the contents of a folder named 'matlab'. On the right, a script editor window shows a large array of numbers, likely representing a state vector or a list of nodes. The numbers are arranged in rows and columns, with some rows starting with indices like 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 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, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 1010, 1011, 1012, 1013, 1014, 1015, 1016, 1017, 1018, 1019, 1020, 1021, 1022, 1023, 1024, 1025, 1026, 1027, 1028, 1029, 1030, 1031, 1032, 1033, 1034, 1035, 1036, 1037, 1038, 1039, 1040, 1041, 1042, 1043, 1044, 1045, 1046, 1047, 1048, 1049, 1050, 1051, 1052, 1053, 1054, 1055, 1056, 1057, 1058, 1059, 1060, 1061, 1062, 1063, 1064, 1065, 1066, 1067, 1068, 1069, 1070, 1071, 1072, 1073, 1074, 1075, 1076, 1077, 1078, 1079, 1080, 1081, 1082, 1083, 1084, 1085, 1086, 1087, 1088, 1089, 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1097, 1098, 1099, 1100, 1101, 1102, 1103, 1104, 1105, 1106, 1107, 1108, 1109, 1110, 1111, 1112, 1113, 1114, 1115, 1116, 1117, 1118, 1119, 1120, 1121, 1122, 1123, 1124, 1125, 1126, 1127, 1128, 1129, 1130, 1131, 1132, 1133, 1134, 1135, 1136, 1137, 1138, 1139, 1140, 1141, 1142, 1143, 1144, 1145, 1146, 1147, 1148, 1149, 1150, 1151, 1152, 1153, 1154, 1155, 1156, 1157, 1158, 1159, 1160, 1161, 1162, 1163, 1164, 1165, 1166, 1167, 1168, 1169, 1170, 1171, 1172, 1173, 1174, 1175, 1176, 1177, 1178, 1179, 1180, 1181, 1182, 1183, 1184, 1185, 1186, 1187, 1188, 1189, 1190, 1191, 1192, 1193, 1194, 1195, 1196, 1197, 1198, 1199, 1200, 1201, 1202, 1203, 1204, 1205, 1206, 1207, 1208, 1209, 1210, 1211, 1212, 1213, 1214, 1215, 1216, 1217, 1218, 1219, 1220, 1221, 1222, 1223, 1224, 1225, 1226, 1227, 1228, 1229, 1230, 1231, 1232, 1233, 1234, 1235, 1236, 1237, 1238, 1239, 1240, 1241, 1242, 1243, 1244, 1245, 1246, 1247, 1248, 1249, 1250, 1251, 1252, 1253, 1254, 1255, 1256, 1257, 1258, 1259, 1260, 1261, 1262, 1263, 1264, 1265, 1266, 1267, 1268, 1269, 1270, 1271, 1272, 1273, 1274, 1275, 1276, 1277, 1278, 1279, 1280, 1281, 1282, 1283, 1284, 1285, 1286, 1287, 1288, 1289, 1290, 1291, 1292, 1293, 1294, 1295, 1296, 1297, 1298, 1299, 1300, 1301, 1302, 1303, 1304, 1305, 1306, 1307, 1308, 1309, 1310, 1311, 1312, 1313, 1314, 1315, 1316, 1317, 1318, 1319, 1320, 1321, 1322, 1323, 1324, 1325, 1326, 1327, 1328, 1329, 1330, 1331, 1332, 1333, 1334, 1335, 1336, 1337, 1338, 1339, 1340, 1341, 1342, 1343, 1344, 1345, 1346, 1347, 1348, 1349, 1350, 1351, 1352, 1353, 1354, 1355, 1356, 1357, 1358, 1359, 1360, 1361, 1362, 1363, 1364, 1365, 1366, 1367, 1368, 1369, 1370, 1371, 1372, 1373, 1374, 1375, 1376, 1377, 1378, 1379, 1380, 1381, 1382, 1383, 1384, 1385, 1386, 1387, 1388, 1389, 1390, 1391, 1392, 1393, 1394, 1395, 1396, 1397, 1398, 1399, 1400, 1401, 1402, 1403, 1404, 1405, 1406, 1407, 1408, 1409, 1410, 1411, 1412, 1413, 1414, 1415, 1416, 1417, 1418, 1419, 1420, 1421, 1422, 1423, 1424, 1425, 1426, 1427, 1428, 1429, 1430, 1431, 1432, 1433, 1434, 1435, 1436, 1437, 1438, 1439, 1440, 1441, 1442, 1443, 1444, 1445, 1446, 1447, 1448, 1449, 1450, 1451, 1452, 1453, 1454, 1455, 1456, 1457, 1458, 1459, 1460, 1461, 1462, 1463, 1464, 1465, 1466, 1467, 1468, 1469, 1470, 1471, 1472, 1473, 1474, 1475, 1476, 1477, 1478, 1479, 1480, 1481, 1482, 1483, 1484, 1485, 1486, 1487, 1488, 1489, 1490, 1491, 1492, 1493, 1494, 1495, 1496, 1497, 1498, 1499, 1500, 1501, 1502, 1503, 1504, 1505, 1506, 1507, 1508, 1509, 1510, 1511, 1512, 1513, 1514, 1515, 1516, 1517, 1518, 1519, 1520, 1521, 1522, 1523, 1524, 1525, 1526, 1527, 1528, 1529, 1530, 1531, 1532, 1533, 1534, 1535, 1536, 1537, 1538, 1539, 1540, 1541, 1542, 1543, 1544, 1545, 1546, 1547, 1548, 1549, 1550, 1551, 1552, 1553, 1554, 1555, 1556, 1557, 1558, 1559, 1560, 1561, 1562, 1563, 1564, 1565, 1566, 1567, 1568, 1569, 1570, 1571, 1572, 1573, 1574, 1575, 1576, 1577, 1578, 1579, 1580, 1581, 1582, 1583, 1584, 1585, 1586, 1587, 1588, 1589, 1590, 1591, 1592, 1593, 1594, 1595, 1596, 1597, 1598, 1599, 1600, 1601, 1602, 1603, 1604, 1605, 1606, 1607, 1608, 1609, 1610, 1611, 1612, 1613, 1614, 1615, 1616, 1617, 1618, 1619, 1620, 1621, 1622, 1623, 1624, 1625, 1626, 1627, 1628, 1629, 1630, 1631, 1632, 1633, 1634, 1635, 1636, 1637, 1638, 1639, 1640, 1641, 1642, 1643, 1644, 1645, 1646, 1647, 1648, 1649, 1650, 1651, 1652, 1653, 1654, 1655, 1656, 1657, 1658, 1659, 1660, 1661, 1662, 1663, 1664, 1665, 1666, 1667, 1668, 1669, 1670, 1671, 1672, 1673, 1674, 1675, 1676, 1677, 1678, 1679, 1680, 1681, 1682, 1683, 1684, 1685, 1686, 1687, 1688, 1689, 1690, 1691, 1692, 1693, 1694, 1695, 1696, 1697, 1698, 1699, 1700, 1701, 1702, 1703, 1704, 1705, 1706, 1707, 1708, 1709, 1710, 1711, 1712, 1713, 1714, 1715, 1716, 1717, 1718, 1719, 1720, 1721, 1722, 1723, 1724, 1725, 1726, 1727, 1728, 1729, 1730, 1731, 1732, 1733, 1734, 1735, 1736, 1737, 1738, 1739, 1740, 1741, 1742, 1743, 1744, 1745, 1746, 1747, 1748, 1749, 1750, 1751, 1752, 1753, 1754, 1755, 1756, 1757, 1758, 1759, 1760, 1761, 1762, 1763, 1764, 1765, 1766, 1767, 1768, 1769, 1770, 1771, 1772, 1773, 1774, 1775, 1776, 1777, 1778, 1779, 1780, 1781, 1782, 1783, 1784, 1785, 1786, 1787, 1788, 1789, 1790, 1791, 1792, 1793, 1794, 1795, 1796, 1797, 1798, 1799, 1800, 1801, 1802, 1803, 1804, 1805, 1806, 1807, 1808, 1809, 1810, 1811, 1812, 1813, 1814, 1815, 1816, 1817, 1818, 1819, 1820, 1821, 1822, 1823, 1824, 1825, 1826, 1827, 1828, 1829, 1830, 1831, 1832, 1833, 1834, 1835, 1836, 1837, 1838, 1839, 1840, 1841, 1842, 1843, 1844, 1845, 1846, 1847, 1848, 1849, 1850, 1851, 1852, 1853, 1854, 1855, 1856, 1857, 1858, 1859, 1860, 1861, 1862, 1863, 1864, 1865, 1866, 1867, 1868, 1869, 1870, 1871, 1872, 1873, 1874, 1875, 1876, 1877, 1878, 1879, 1880, 1881, 1882, 1883, 1884, 1885, 1886, 1887, 1888, 1889, 1890, 1891, 1892, 1893, 1894, 1895, 1896, 1897, 1898, 1899, 1900, 1901, 1902, 1903, 1904, 1905, 1906, 1907, 1908, 1909, 1910, 1911, 1912, 1913, 1914, 1915, 1916, 1917, 1918, 1919, 1920, 1921, 1922, 1923, 1924, 1925, 1926, 1927, 1928, 1929, 1930, 1931, 1932, 1933, 1934, 1935, 1936, 1937, 1938, 1939, 1940, 1941, 1942, 1943, 1944, 1945, 1946, 1947, 1948, 1949, 1950, 1951, 1952, 1953, 1954, 1955, 1956, 1957, 1958, 1959, 1960, 1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, 1971, 1972, 1973, 1974, 1975, 1976, 1977, 1978, 1979, 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063, 2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074, 2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084, 2085, 2086, 2087, 2088, 2089, 2090, 2091, 2092, 2093, 2094, 2095, 2096, 2097, 2098, 2099, 2100, 2101, 2102, 2103, 2104, 2105, 2106, 2107, 2108, 2109, 2110, 2111, 2112, 2113, 2114, 2115, 2116, 2117, 2118, 2119, 2120, 2121, 2122, 2123, 2124, 2125, 2126, 2127, 2128, 2129, 2130, 2131, 2132, 2133, 2134, 2135, 2136, 2137, 2138, 2139, 2140, 2141, 2142, 2143, 2144, 2145, 2146, 2147, 2148, 2149, 2150, 2151, 2152, 2153, 2154, 2155, 2156, 2157, 2158, 2159, 2160, 2161, 2162, 2163, 2164, 2165, 2166, 2167, 2168,



```
42 F = zeros(1,1,1); % Message position (1,1,1)
43 Message position (1,1,1)
44 msg = randi(0,1,1,1); % Generate random k-bit message
45
46 n = zeros(1,1);
47
48 v(1:1:1) = msg; % msg k-bit message bits
49
50
51 N = 1; % Number of bits combined
52 for d = 1:1:1
53     for i = 1:1:1
54         a = zeros(1,1); % First part
55         b = zeros(1,1); % Second part
56         v(1:1:1) = [a;b]; % Concatenating
57     end
58     N = N * 2;
59 end
60 cword = v;
61
62
63
64
65
66
67
68
```



PROF. ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 1



```
1 % Parameters
2 Nbits = 10;
3 max = 4;
4 msg = 1;
5 msg = 1;
6 offset = 2;
7
8 % Load base matrices N_b_14_141
9 N = N_b_1_14;
10 [N0,N1] = size(N);
11 z = 14;
12
13 % Size = numBits--> Number of non-1 in k
14 % Log = zeroMatrix(z, z, -1, 2); % Logspace storage for a1000
15
16 k = (N0-N1)/2; % Number of message bits
17 n = N0/2; % Number of codeword bits
18
19 Rate = k/(N0-2);
20 SNR = 10*log10(1);
21 alpha = sqrt(1/(2*Nbits*SNR));
22
23 NIterations = 10; % Number of iterations = 100
24 for i = 1:NIterations
25     msg = randi(0,1,1); % Generate random k-bit message
26     msg = zeros(1,1); % All-zero message
27     % Encoding
28     cword = zeros(1,1); % All-zero codeword
29
30     s = 1 - 2 * cword; % Convert bit to symbol conversion
31     r = s + alpha * randi(0,1,1); % AWGN channel
32     % Structuring of message
33     r(1:1:1) = msg;
34
35
```



PROF. ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 1

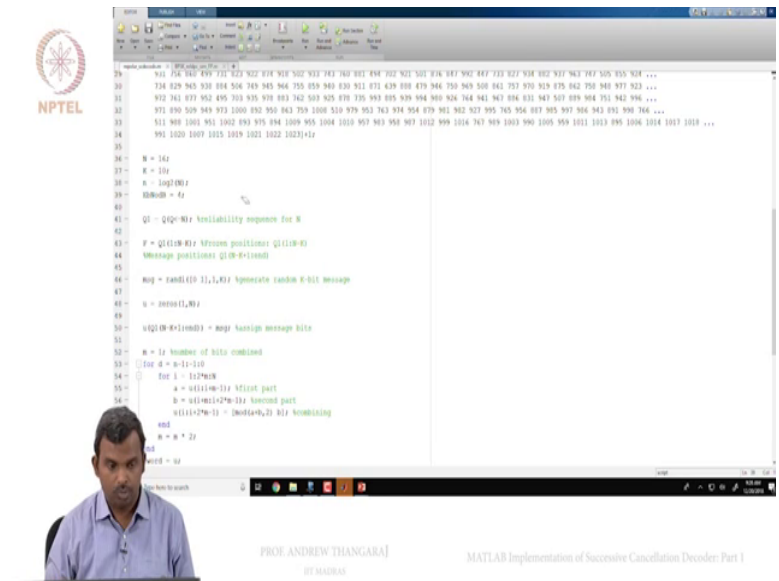


```
1
2 % Load base matrices N_b_14_141
3 N = N_b_1_14;
4 [N0,N1] = size(N);
5 z = 14;
6
7 % Size = numBits--> Number of non-1 in k
8 % Log = zeroMatrix(z, z, -1, 2); % Logspace storage for a1000
9
10 k = (N0-N1)/2; % Number of message bits
11 n = N0/2; % Number of codeword bits
12
13 SNR = 10*log10(1);
14 alpha = sqrt(1/(2*Nbits*SNR));
15
16 NIterations = 10; % Number of iterations = 100
17 for i = 1:NIterations
18     for l = 1:Nbits
19         msg = randi(0,1,1); % Generate random k-bit message
20         msg = zeros(1,1); % All-zero message
21         % Encoding
22         cword = zeros(1,1); % All-zero codeword
23
24         s = 1 - 2 * cword; % Convert bit to symbol conversion
25         r = s + alpha * randi(0,1,1); % AWGN channel
26         % Structuring of message
27         r(1:1:1) = msg;
28         % Initialization
29         pq = floor(r/max(msg));
30         q(1:1:1) = msg;
31         % Decoding
32         r(1:1:1) = r - alpha*q;
33
34
```



PROF. ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 1



So what a MATLAB, so this is a the encoder that we had before, it had the reliability sequence, you remember the encoder, so what I am going to do is, I am going to save this as a decode, so I call it SE decode just to distinguish from other thing, so will have the reliability sequence, will keep it, we keep N as something small just so that we can do some debugging, maybe keep this as 10, we get the so frozen positions, so Q1 the frozen positions it is good to store the frozen positions explicitly.

So I will store them as F equals Q1 of 1 colon N minus K okay, so because in the decoder you to check if a particular index is frozen or not okay, so for that will keep this F as the list of frozen positions and after that the message and the U and the encoding can happen as before, so right now you finish with the encoding U is the code word and you can transmit okay.

So let us transmit, for transmitting let just steal, so C words equals U, so for transmitting and all that setup let steal something from here, so will steal the EB over N not which has to be set up somewhere here, so 0 DB I keep it here, 4 DB, then you have this usual computations of rate EB over N not and this I have to change the rate calculation, what is it for us? Rate for us is K by N okay.

Node position if you remember I just wrote it down here $2 \text{ par } D \text{ minus } 1 \text{ plus } 1$ okay, it is $2^{\text{power depth minus } 1 \text{ plus node}}$, and $N \text{ plus } 1$, why do I do $\text{plus } 1$ that is the MATLAB adjustment for the position of node in node state better right, so that is the $N \text{ Poss}$ and then you have to do multiple things depending on whether or not $N \text{ Poss}$ is a, what is the state okay, so you can do a lot of if conditions here, it just few states only, if $NS \text{ of } N \text{ Poss}$ equal to equal to 0 okay, so that is one condition else so for now I simply say display work progress will write it, else if, so it is good to end this else is, if $NS \text{ of } N \text{ Poss}$, why is it?

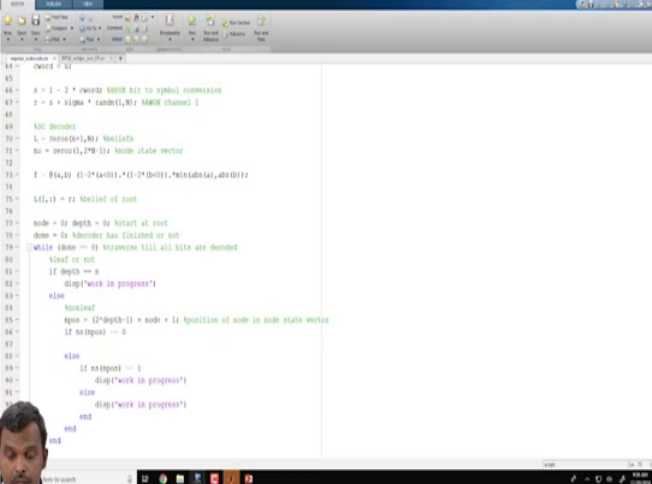

Okay, so I think I made a mistake here, there should be round bracket, $NS \text{ of } N \text{ Poss}$ equal to equal to 1, you have to do something else if $NS \text{ of } N \text{ Poss}$ equal to, actually else you do not need, means if it is not 0 or 1, it is going to be 2 and you know what to do, for now I simply say display work in progress, whenever it finishes, will finish okay alright.

So depending on where we are, we do different things that is good okay, then okay, so that is nice alright, so even before all of this I think one thing you can do is, you can check whether your leaf or not right, if you are the leaf than you know what to do, so this is assuming non-leaf okay.


So if your leaf, there is a very simple thing to do here, so we will put that code in a little bit later so for now I am assuming, I am non-leaf okay, I am proceeding to do that part, if you leafs, so you have to check that, so how to check that? If $\text{depth equal to equal to } N$ okay, so if you have reached the last one then you do something else right, so only if you are not so you to do this non-leaf stuff right.

So let us push this to the right and then get N here so that things work out, so I will just keep work in progress here so that we come back and read something there okay, so hopefully this is clear, so how does that operation works, so in every node I first check if I am leaf or not, if I am not leaf I have to do something, if I am leaf I just have to do you know, I have to make a decision and then do something, only if I am not leaf I do anything okay.

(Refer Slide Time: 28:37)

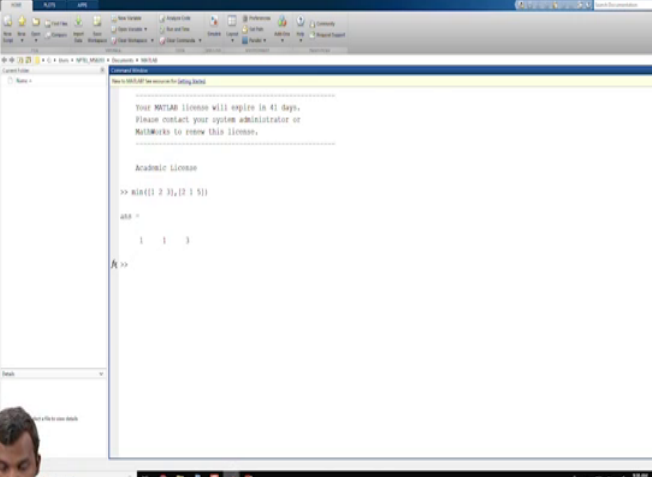



```
44 - clear; %
45 -
46 - s = 1 - 2 * (word2 ASCII bit to symbol conversion
47 - z = s + sigma * randi(2,1) * NAWW channel 1
48 -
49 - %%% decoder
50 - L = (word2(1,N) * Nbits)
51 - nu = zeros(1,2^L) % node state vector
52 -
53 - r = [z, z * (-1)^(1+1:1:1) * (-1)^(2+1:1:1) * (-1)^(3+1:1:1) * (-1)^(4+1:1:1)]; % received signal
54 -
55 - L1(1) = r; % initial of root
56 -
57 - node = 0; depth = 0; % start at root
58 - done = 0; % decoder has finished or not
59 - while (done == 0) % iterate till all bits are decoded
60 -     kind = 0;
61 -     if depth == L
62 -         disp('work in progress')
63 -     else
64 -         %next
65 -         npos = 2^depth - 1 + node + 1; % position of node in node state vector
66 -         if npos == 0
67 -
68 -         else
69 -             if npos == 1
70 -                 disp('work in progress')
71 -             else
72 -                 disp('work in progress')
73 -             end
74 -         end
75 -     end
76 - end
```




PROF. ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 1

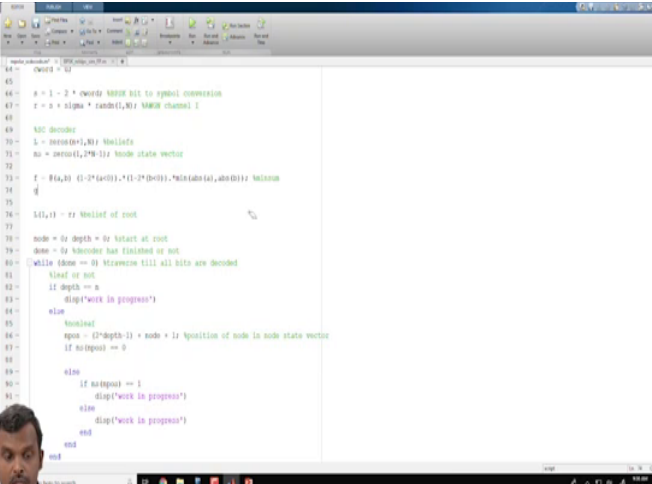



```
%%
>> bit(1:2,3),(2:1,5)
ans =
     1     1     3
>>
```




PROF. ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 1


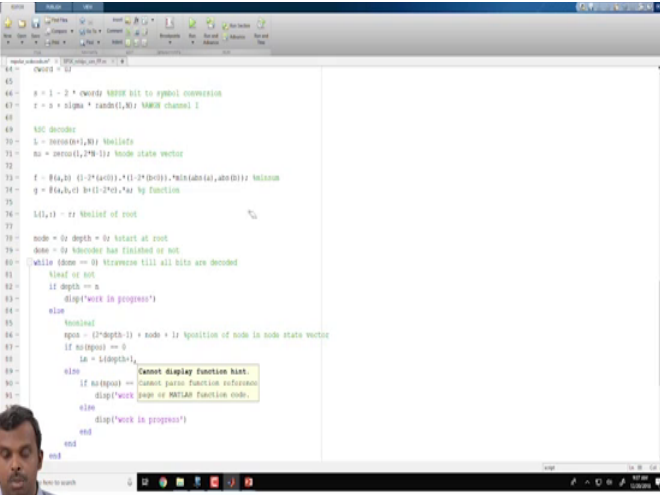


```
44 - clear; %
45 -
46 - s = 1 - 2 * (word2 ASCII bit to symbol conversion
47 - z = s + sigma * randi(2,1) * NAWW channel 1
48 -
49 - %%% decoder
50 - L = (word2(1,N) * Nbits)
51 - nu = zeros(1,2^L) % node state vector
52 -
53 - r = [z, z * (-1)^(1+1:1:1) * (-1)^(2+1:1:1) * (-1)^(3+1:1:1) * (-1)^(4+1:1:1)]; % received signal
54 -
55 - L1(1) = r; % initial of root
56 -
57 - node = 0; depth = 0; % start at root
58 - done = 0; % decoder has finished or not
59 - while (done == 0) % iterate till all bits are decoded
60 -     kind = 0;
61 -     if depth == L
62 -         disp('work in progress')
63 -     else
64 -         %next
65 -         npos = 2^depth - 1 + node + 1; % position of node in node state vector
66 -         if npos == 0
67 -
68 -         else
69 -             if npos == 1
70 -                 disp('work in progress')
71 -             else
72 -                 disp('work in progress')
73 -             end
74 -         end
75 -     end
76 - end
```




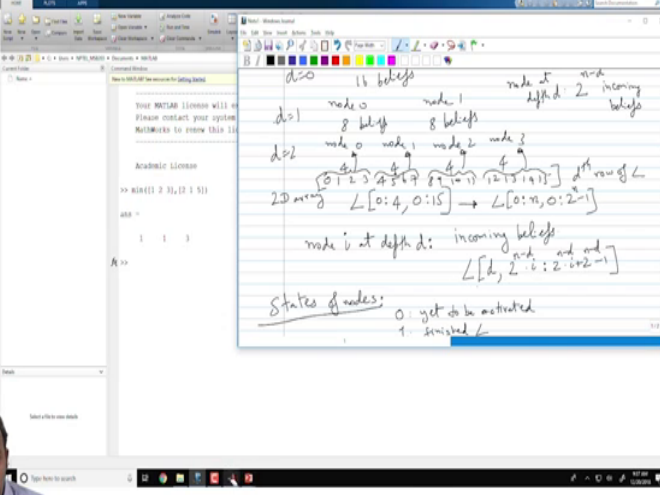
PROF. ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 1

PROF. ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 1

PROF. ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 1

So let us start on this, so if I am not leaf I am in state 0, let us see how to write this up okay, so what do I do? I have to do minsum and for minsum we have this simple small little F function and the G function right, so one of the things I like to do is to write this F and G functions as this MATLAB in-line functions.

So we can write F as AB, so I do not want to use the MATLAB sign function, the problem with MATLAB sign function is it gives the 0 for 0, so that is not very nice so maybe we can do this modified thing right, so we can say 1 minus 2 times A less than 0 right, so if there is greater than or equal to 0, now maybe I should put here A less than 0, yes this is good because N if A is negative this is going to be minus 1, if a 0 or positive this is going to be plus 1 okay.

So this times 1 minus 2 times, so it is goes to assume that A and B will be vectors, so I will put a dot product here and then dot into min of ABS of A, ABS of B, hopefully this will work,

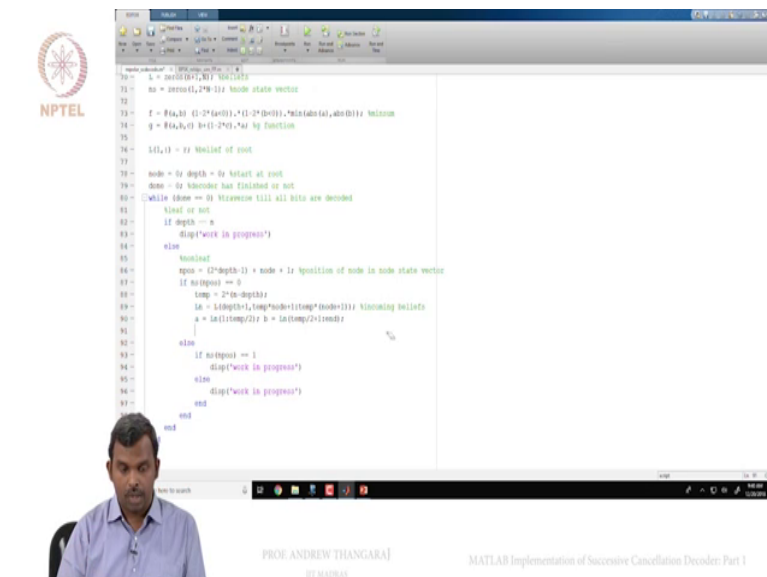
this cut okay, so this looks okay enough for me sometimes I like to check if this will work min of for instance 1, 2, 3 2, 1, 5 hopefully will works okay, it is good to check once in a while that what you are writing is correct.

So this is F okay, so this is F function if you remember, it does the minsum okay, general vector also it will work, then I will also need the G, G has three inputs ABC and its output is considerably simpler, it is B plus 1 minus 2 into C, dot into A, this is the G function okay, so you remember the G function it does, R2 minus R1, R2 plus R1 depending on whether or not C is 0 or 1 okay, so that is G and F okay.

So once you write G and F, this is easy enough to write down, so we just have to figure out what A is and what B is? Right, so what is A? Remember for node and depth, we know what is the corresponding L value, so L node equals L of depth plus 1, Right you remember the little formula that I wrote down.

So its 2 par N minus D times I from then 2 par N minus D times I plus this guy right, so this 2 par N minus D keeps appearing quite prominently, so it is probably worth computing that first okay, so I call it N equals 2 power N minus depth okay and then this is going to be temp times node plus 1 colon temp times node plus temp so it is just little crop the node plus 1 okay.

(Refer Slide Time: 32:47)



```
50 = k = zeros(1,2^N); % node state vector
51 = m = zeros(1,2^N); % node state vector
52 = f = f(a,b) (1-2^a+1)*(1-2^b+1); % minsum(a,b) by function
53 = g = g(a,b,c) (1-2^a+1)*(1-2^b+1); % by function
54 = L(1,1) = 1; % initial of root
55 =
56 = node = 0; depth = 0; % start at root
57 = done = 0; % whether has finished or not
58 = while (done == 0) % stream till all bits are decoded
59 = % load or not
60 = if depth == n
61 = disp('work in progress');
62 = else
63 = % minsum
64 = npos = (2^depth-1) + node + 1; % position of node in node state vector
65 = if notpos == 0
66 = temp = 2^(n-depth);
67 = [a = 1:length(1,temp*node+1:temp*node+1); % minsum(a,b)
68 = a = 1:(1+temp/2); b = 1:(1+temp/2); % minsum(a,b)
69 = ]
70 = if notpos == 1
71 = disp('work in progress');
72 = else
73 = disp('work in progress');
74 = end
75 = end
76 = end
```

NPTEL

PROF. ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 1



PROF. ANDREW THANGARAJ
@IITMADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 1

```

70- L = ceil(1/2*(1+sqrt(1+4*beta)));
71- m = zeros(1,2^L); % node state vector
72-
73- f = F(a,b) (1-2^(-L+1))*1-2^(-L+1); %initial(a,b,0); % return
74- g = F(a,b) (1-2^(-L+1))*1-2^(-L+1); % by function
75-
76- L(L,1) = 1; % leaf of root
77-
78- node = 0; % depth = 0; return at root
79- done = 0; % iteration has finished or not
80- while done == 0 % iteration till all bits are decoded
81-   kind = 0; % kind of node
82-   if depth == 0
83-     disp('work in progress')
84-   else
85-     % kind of node
86-     npos = (2^depth-1) + node + 1; % position of node in node state vector
87-     if npos == 0
88-       temp = 2^(L-depth);
89-       Lk = L(depth,1,temp*node+1); % incoming beliefs
90-       a = ceil(temp/2); b = floor(temp/2); % split beliefs into 2
91-       node = node * 2; % depth = depth + 1; % node; % temp; % child
92-       temp = temp / 2; % incoming belief length for left child
93-       L(depth,1,temp*node+1) = f(a,b); % return and storage
94-       npos = 1;
95-     else
96-       if npos == 1
97-         disp('work in progress')
98-       else
99-         disp('work in progress')
100-       end
101-     end
102-   end
103- end

```

So you can do a few charity checks for instance if depth is 0 node will also be 0 okay, so this temp will be just 1 okay, so at depth 0 I get 1, so node is 0 and temp is 1, so it will go from 1 to 1 okay, so thinks that okay because the first time, no I am sorry if depth is 0 temp is 2 power N okay, so it will go from 1 to 2 power N okay.

So I think that is okay temp is 2 power N, node is 0, so it will go from 1 to 2 parent, so it seems fine, so LN is my incoming beliefs okay, so I have pulled it out of the storage, the L is the storage throughout I pulled it out and got LN okay and then I can pull out these A and B or maybe I can write it out explicitly for the next one right, so maybe I should do the A and B, A is LN of, so what is the length of this? The length of this is temp okay.

So LN of 1 colon temp by 2, B is LN of okay, so I have pulled out the corresponding thing, so temp by 2 plus 1 colon end okay, so this is the A and B okay and so we are ready to write down now what the next node should be, node equals, so once, so if it is non-leaf and I am at state 0 I have to proceed to the left child, how do I go to the left child? Node is node into 2, depth is depth plus 1 right, so that is what we do and once you have the node, any node, you have a new temp okay, so you have gone down in-depth, so temp will become temp by 2 right.

So it is 1, one step down okay, right so initially I was at a certain the incoming belief length, the belief length is going down by 2 and then now I can assign what it should go in L? L of depth plus 1, the same thing temp into node plus 1 colon temp into node plus 1 equals F of A, okay that is it, so this was the, this is the next node split beliefs into 2, next node incoming

belief length, next node remember is left child for left child than minsum, minsum and storage right.

So you have to do the minsum calculation and store it back also that is very important, so this is the step for state 0, of course you have to change the state NS of N Poss should become okay as soon I do this NS of N Poss should become 1 okay, so this is done okay, so hopefully you saw how this was done, so I am going to cut this lecture at this point, we will come back and in the next lecture see the other steps and how they work and also we will check how this is working correctly a little bit later but this is a good point to stop this lecture and do the coding in the remaining next lecture.