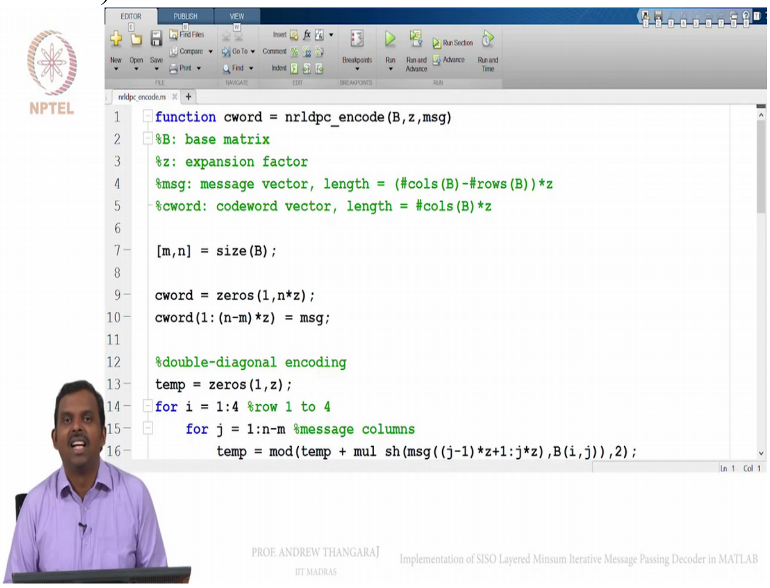
**LDPC and Polar codes in 5G Standard**
**Professor Andrew Thangaraj**
**Department of Electrical Engineering**
**Indian Institute of Technology Madras**
**Implementation of SISO Layered Minsum Iterative Message Passing Decoder in MATLAB**

(Refer Slide Time: 00:16)

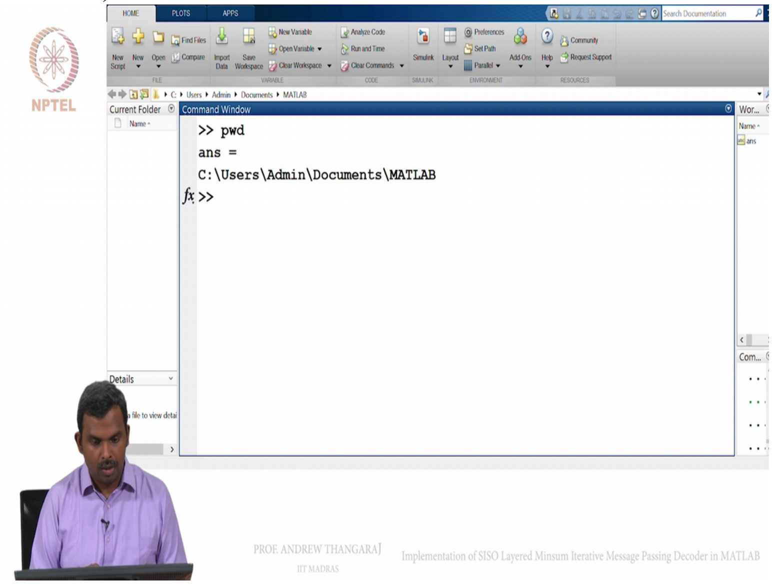



Hello. In this lecture we will start seeing how to write a decoder for LDPC codes in the 5G standard in MATLAB, Ok. So I am going to write a quick and small MATLAB script for doing the decoding. Once again remember that we are not going for very optimized code and very efficient implementation, just to show you how to write a decoder, number 1.

Ok so in the previous MATLAB coding exercise we saw the encoder. I loaded the base matrix and hopefully you still remember all those things. You have the base matrices in a folder and then use; you know how to load it and all that. So let me quickly remind you on how that works.

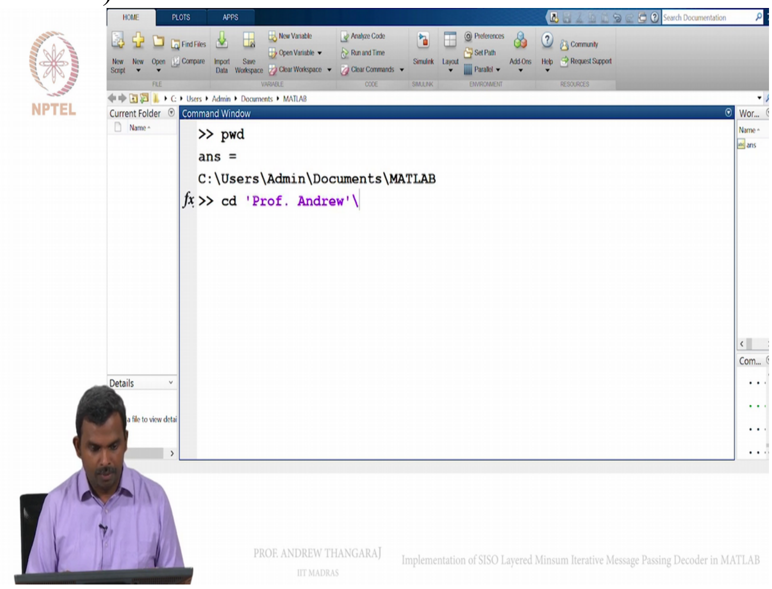
So let me go to MATLAB window.

(Refer Slide Time: 01:06)



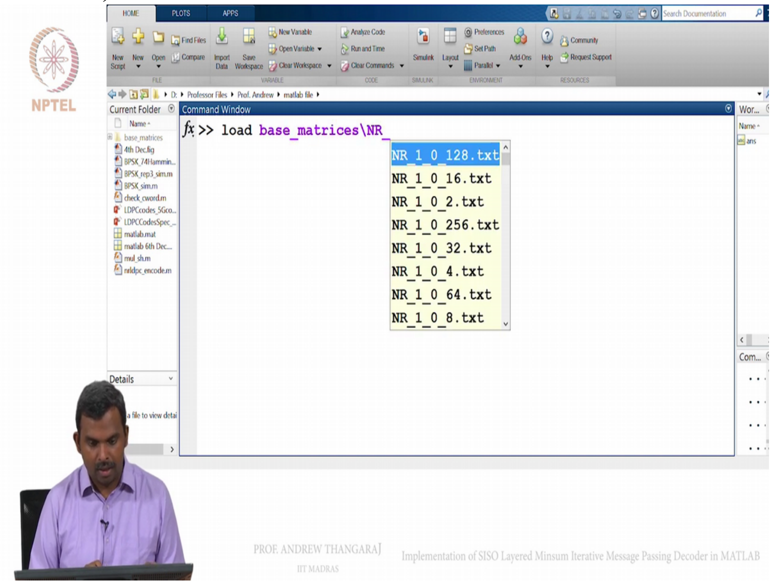

I need to go

(Refer Slide Time: 01:07)




to the correct directory, Ok. So this is my folder in which I do my coding. You can see this folder called base matrices and one can load base matrices N R,

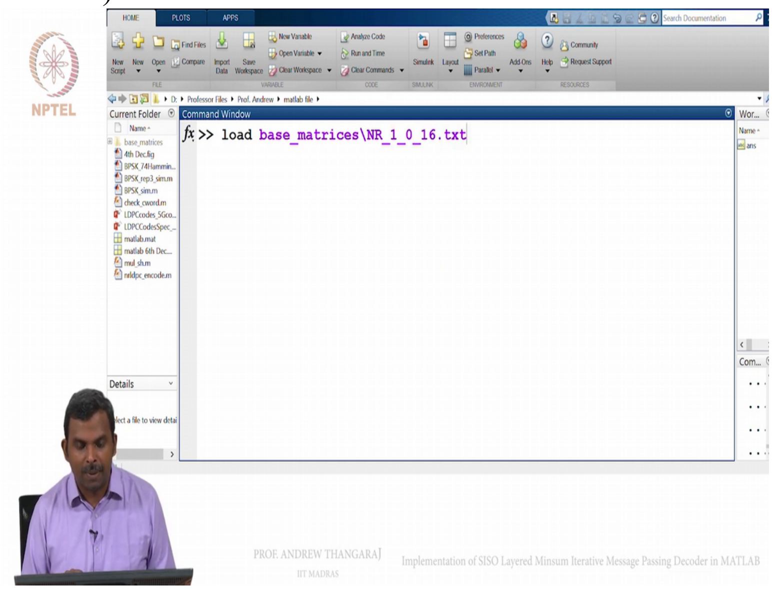
(Refer Slide Time: 01:20)




there are whole bunch of parameters, you remember.

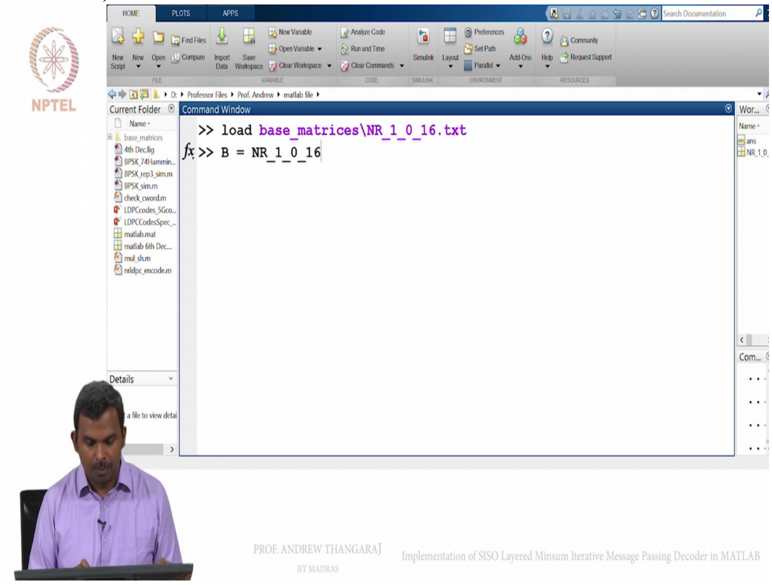
The first one is the base graph number, next one is that j parameter and last one is the expansion factor. So, so let us pick up 1 0 16, you could become anything so we can pick up that.

(Refer Slide Time: 01:36)



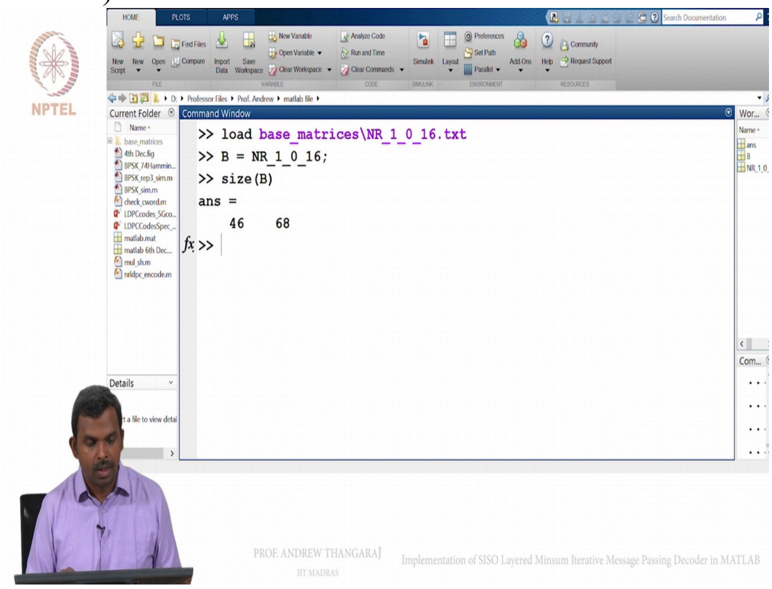

So that is the base matrix and for convenience I like to store it in B,

(Refer Slide Time: 01:44)



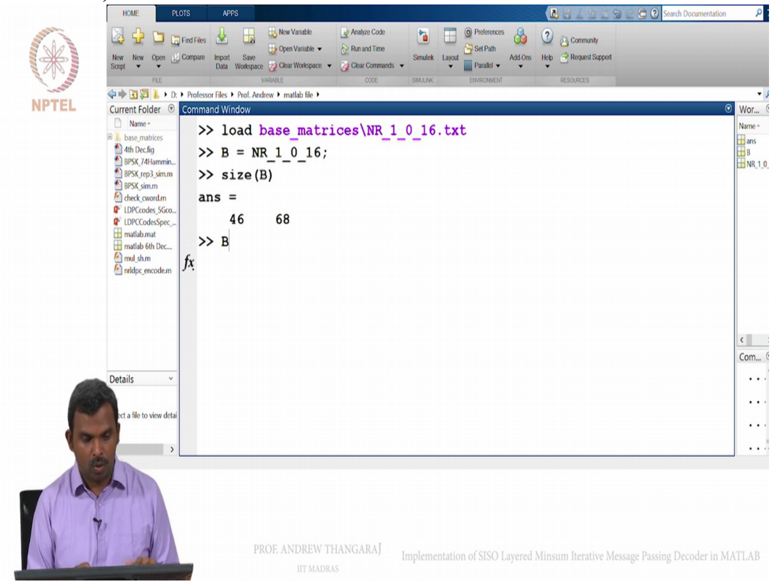

Ok so if you look at size of B,

(Refer Slide Time: 01:47)



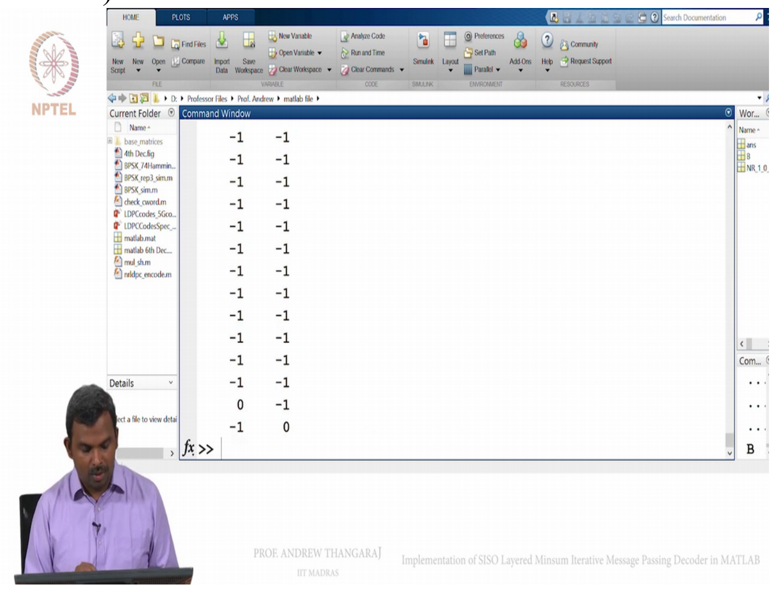

it is 46 by 68. You remember it is,

(Refer Slide Time: 01:50)



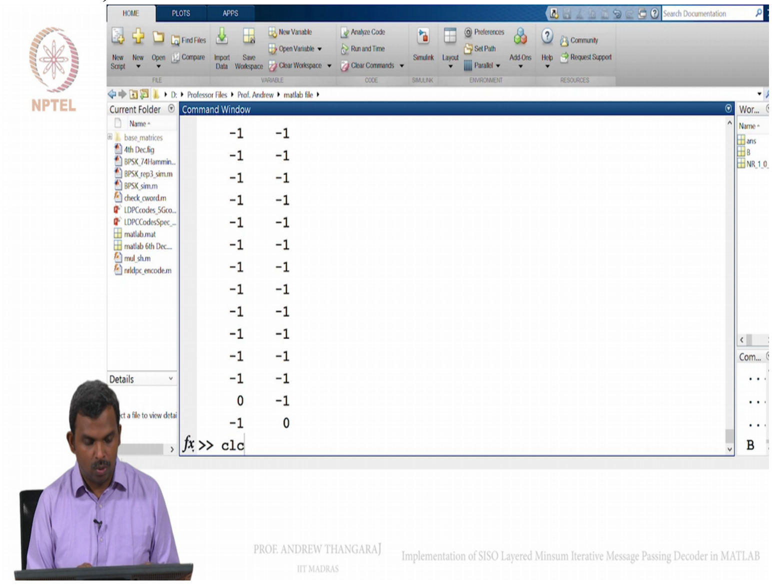

if you just look at the,

(Refer Slide Time: 01:51)



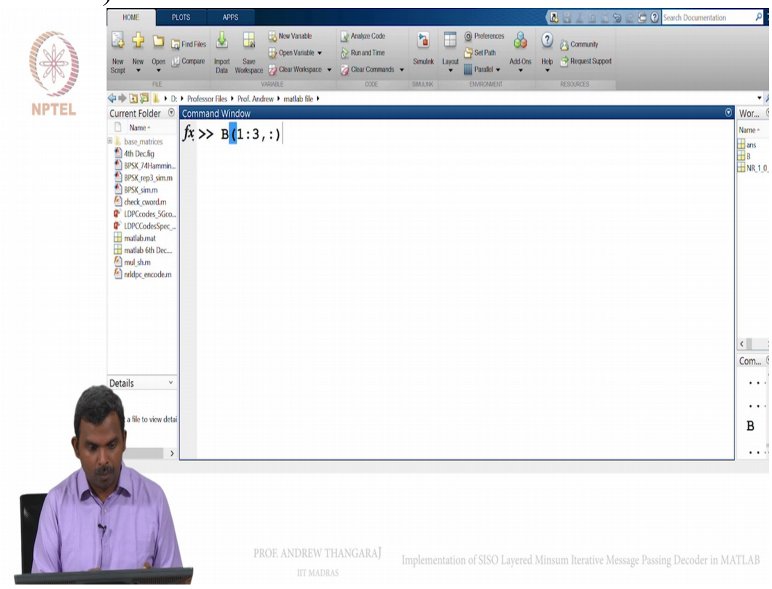

Ok, so

(Refer Slide Time: 01:52)

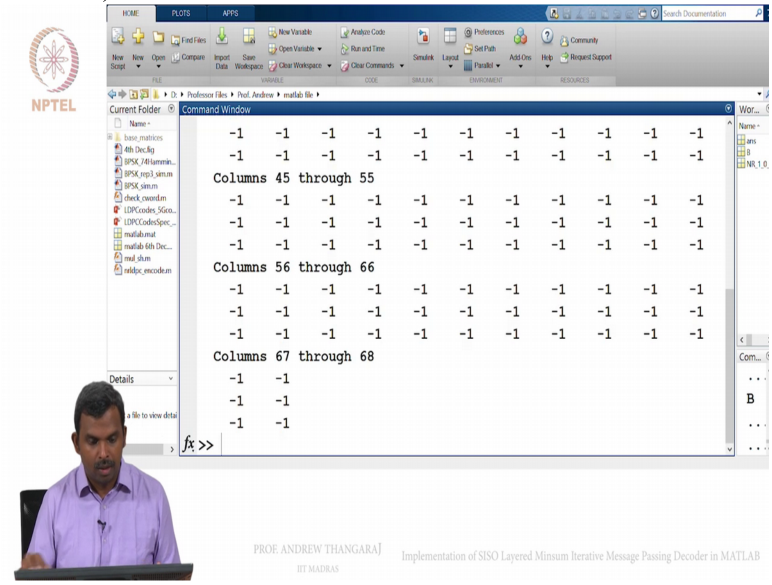



B is little bit to look at, may be you can look at the first three or three rows of B,

(Refer Slide Time: 02:03)

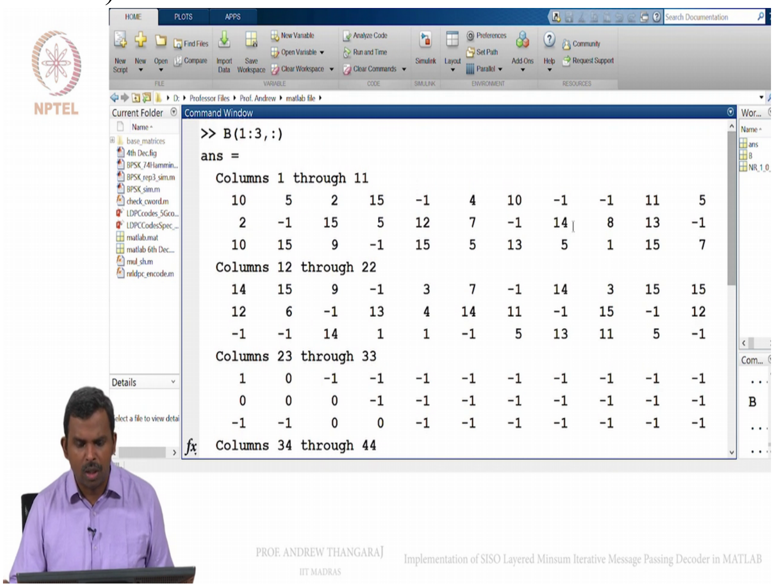


(Refer Slide Time: 02:03)



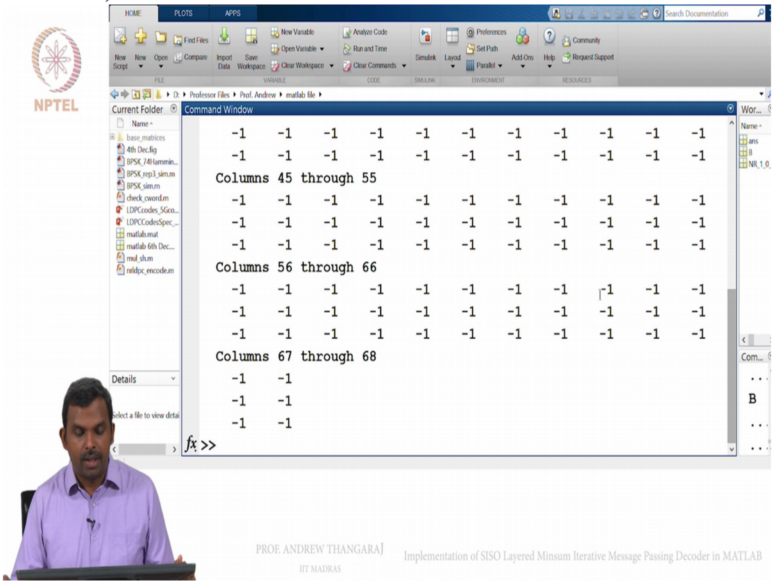

Ok. See whole bunch of minus 1s. So we scroll up a little bit and see the other numbers, Ok. So the

(Refer Slide Time: 02:09)




first few columns have some numbers. Remember the expansion is 16 so you will have numbers from minus 1 to 15 in the base matrix. Then you know how this is, right. So if you look at the number 10, you take the 16 by 16 identity matrix and shift it right by 10 positions, Ok. So this is the, this is the base matrix.

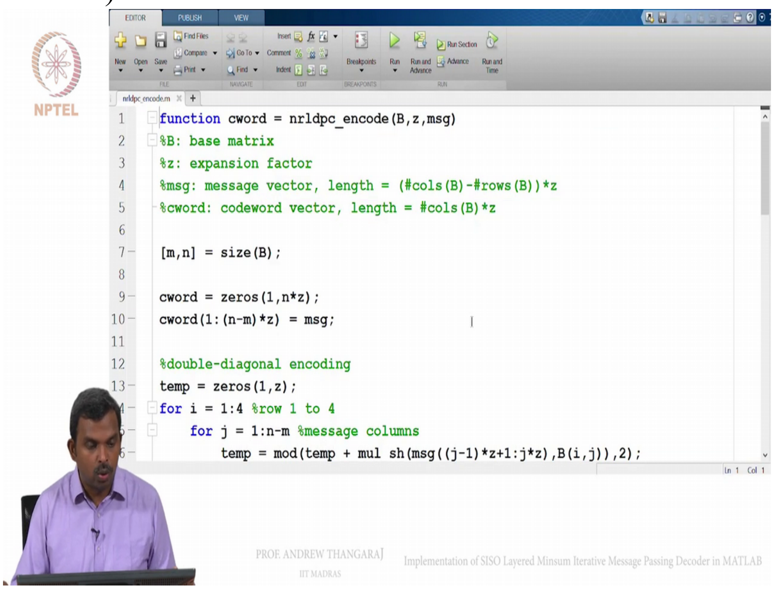
(Refer Slide Time: 02:28)




And this is expanded to form the parity check matrix, Ok.

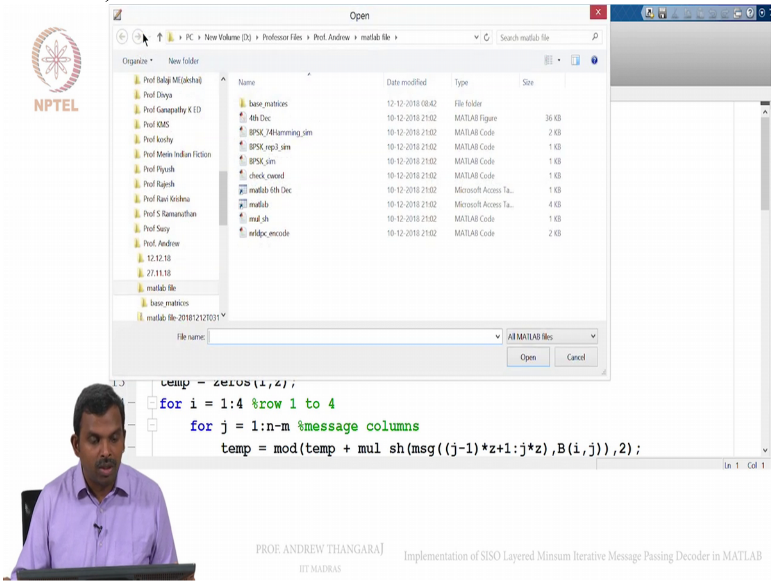
So starting with this base matrix we are going to work and we are going to start building a decoder, Ok. So let us move to the MATLAB editing

(Refer Slide Time: 02:42)



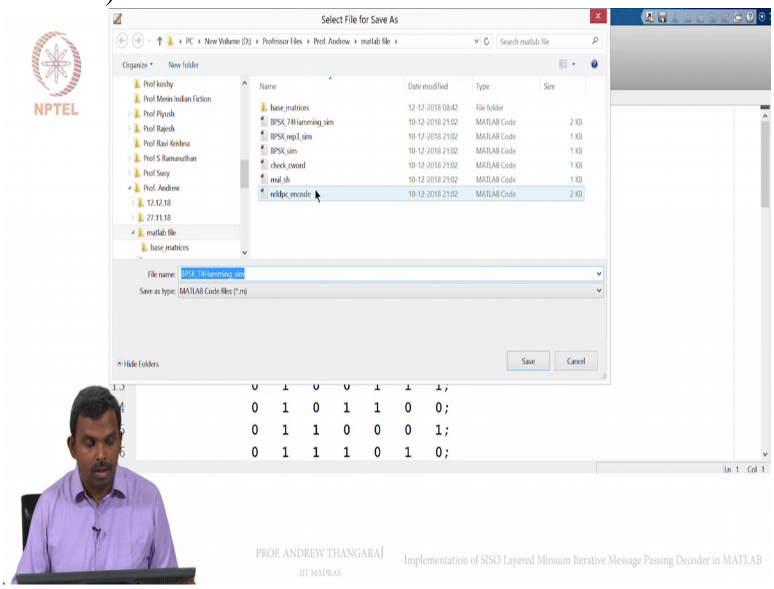

window. This is the encoder, and maybe we need a

(Refer Slide Time: 02:46)




7 4 Hamming code. It is the B P S K 7 4 Hamming code. I am going to save this as

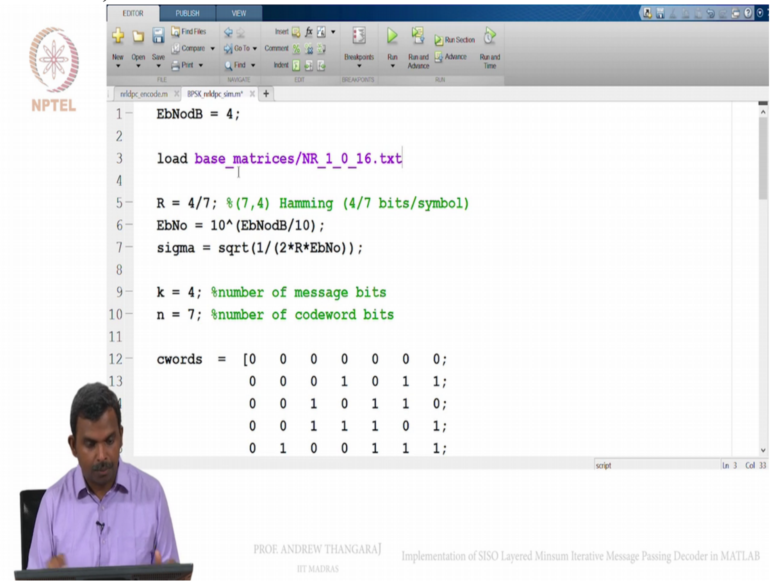
(Refer Slide Time: 02:54)




B P S K n r l d p c sim, Ok.

So we can pick any E b over N naught. I will pick 4, the rate is sort of difficult to determine. So what we can do here is we can repeat what we did before, so we load, base matrix is, so remember I will run it from my correct directory, base matrices and n r 1 0 16, I think it is correct.

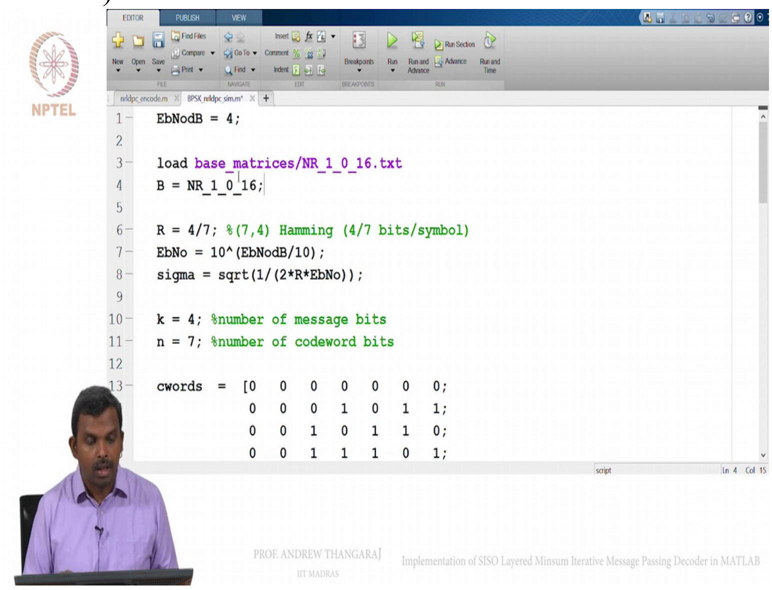
(Refer Slide Time: 03:26)




Let me check that once again.

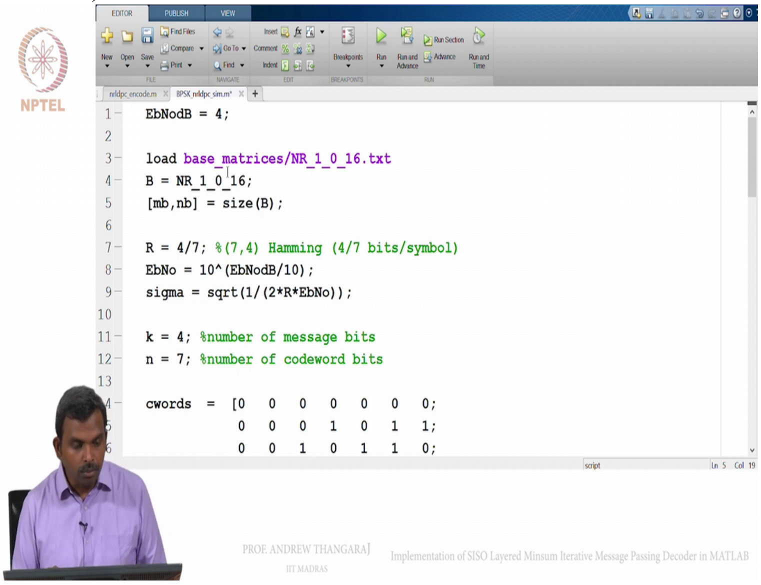
So that is the command we ran, 1 0 16, right so that is the thing. So that is Ok. We loaded that, Ok and then we set B to be equal to 1 0 16. So now I will have my

(Refer Slide Time: 03:46)




base matrix in this B. And now I am ready to find out what m and n are. So I will have, need the size of B. I will use,

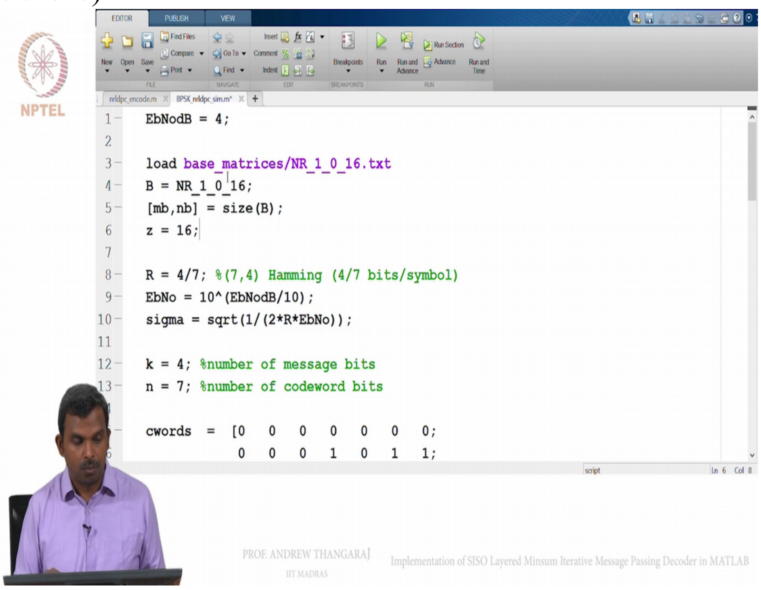
(Refer Slide Time: 04:02)




I will use this.

So we know what m b is going to be, right. m b is going to be 46. n b is going to be 68, right. So that is the N R 1 0 16. If I change it to something else, this will change, Ok. And also clearly z which is the expansion factor

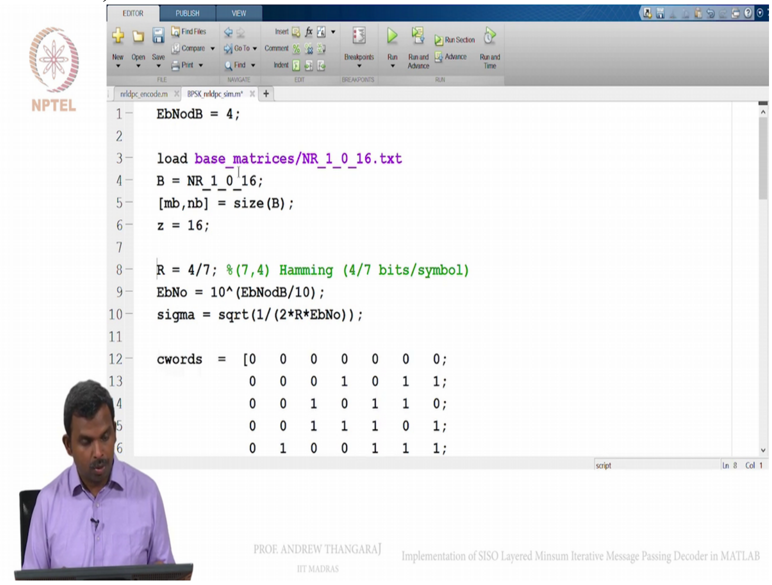
(Refer Slide Time: 04:20)




is 16, Ok.

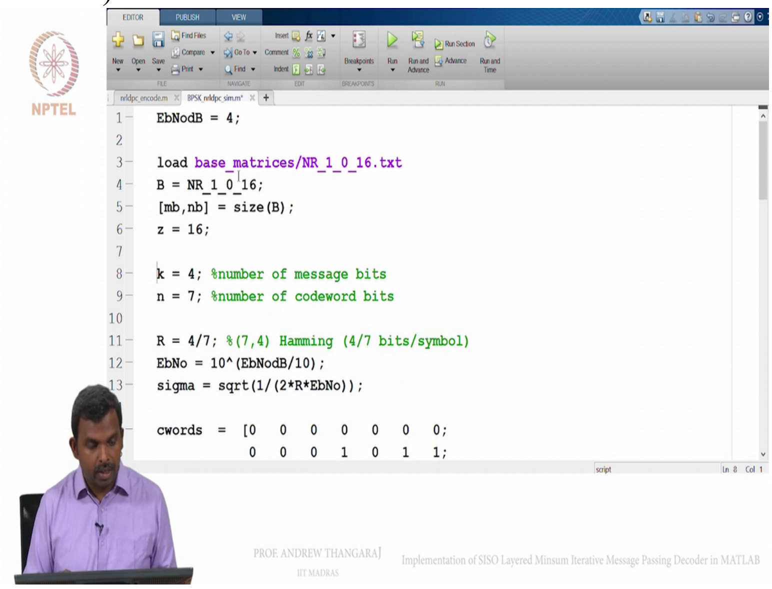
So once I do this I will be able to determine these things.

(Refer Slide Time: 04:29)



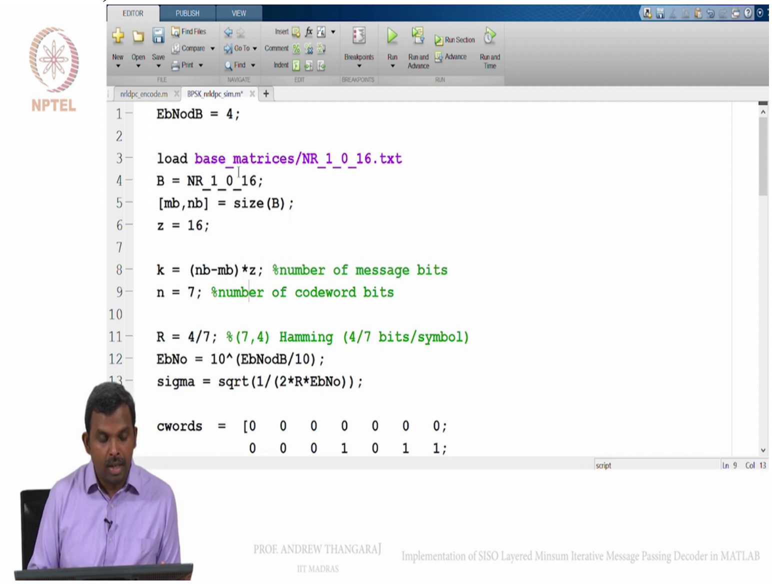

Let me put this little bit forward. k equals

(Refer Slide Time: 04:33)




n b minus m b times z. These are the number of message bits.

(Refer Slide Time: 04:40)



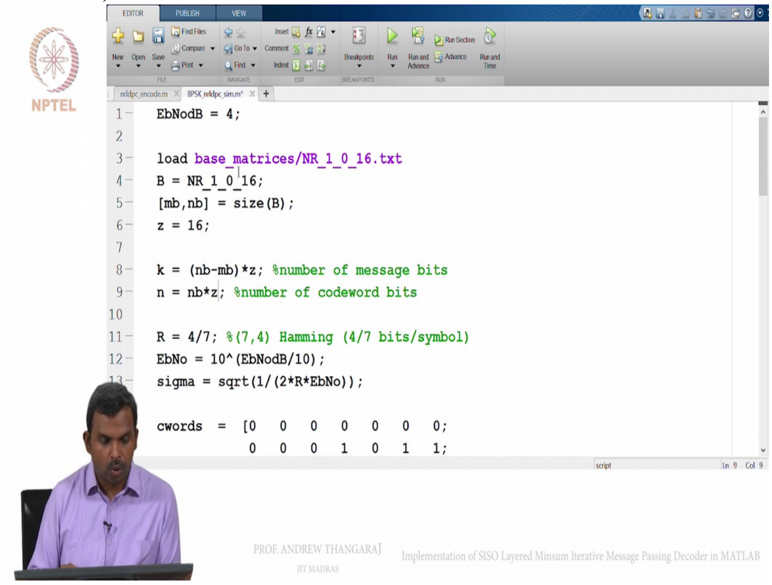

And the number of codeword bits, so usually when you generate the codeword, I am going to generate the entire codeword but and we will simulate that in this code. But typically in, in wireless standards people do something called the rate matching.

So you will do some puncturing, you will do some shortening, combination of these things and put it into some circular buffer, read from that etc. So all of that we are not going to do with this, in this lecture or this class. We will just assume the whole base matrix and the whole parity check matrix is what we simulate, Ok.

So later on, maybe if we have time, I will go through some examples where may be I will show you some puncturing but it is quite easy to program the puncturing part. At least I will mention how that is accommodated, Ok.

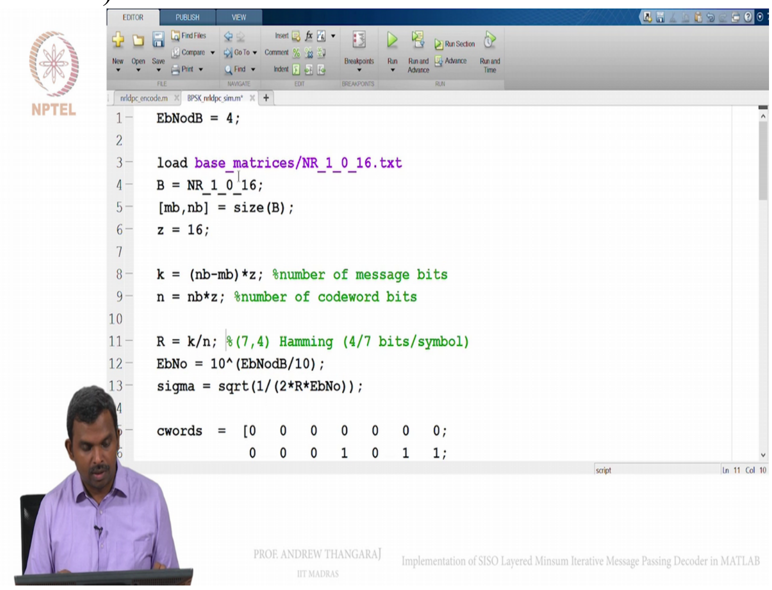
So n as far as I am concerned, is n b times z.

(Refer Slide Time: 05:29)



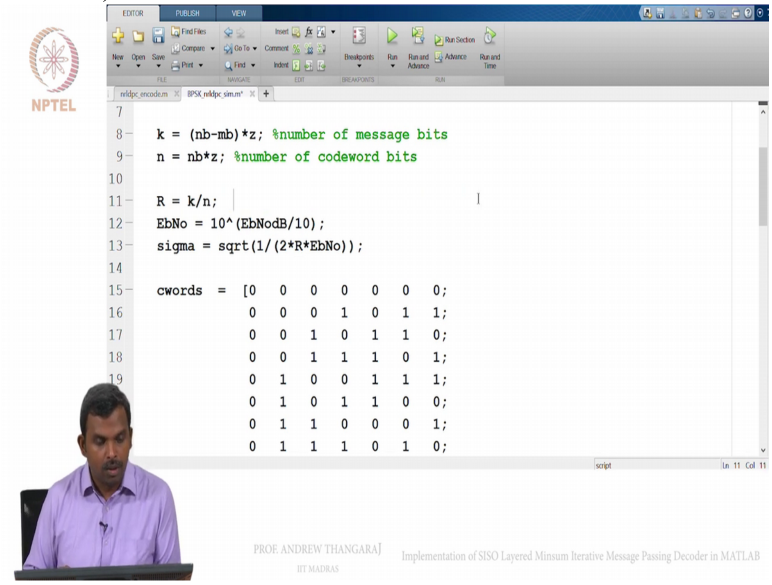

Ok. So remember once again, this is, this is, n b is 46, m b is 68. So the difference is 22. So k will be 22 into 16. And n will be 68 into 16. So that is the current thing I am working, Ok. So the rate is going to be k by n.

(Refer Slide Time: 05:52)




So this is clearly not the Hamming code. So, oops, oops, it is going all over the place. It is not deleted but Ok. Let us just do it. Ok.

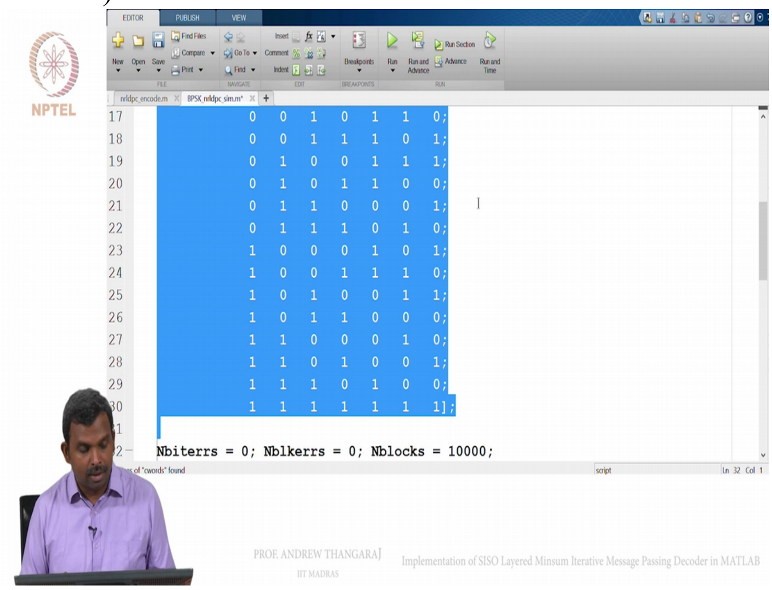
(Refer Slide Time: 06:08)




So once you do this, you have E b over N naught, you have a base matrix, you have the rate and so you can find your sigma, Ok.

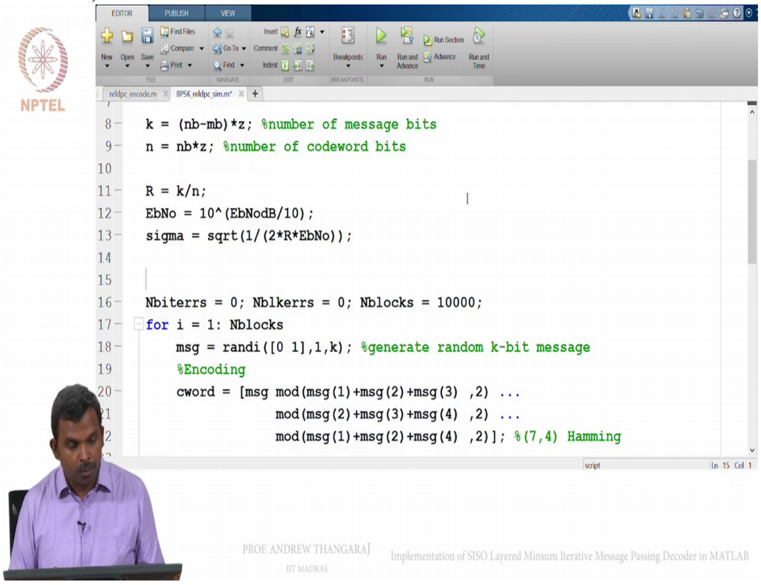
So there is no way I can list out all the codewords. So that part is completely

(Refer Slide Time: 06:19)



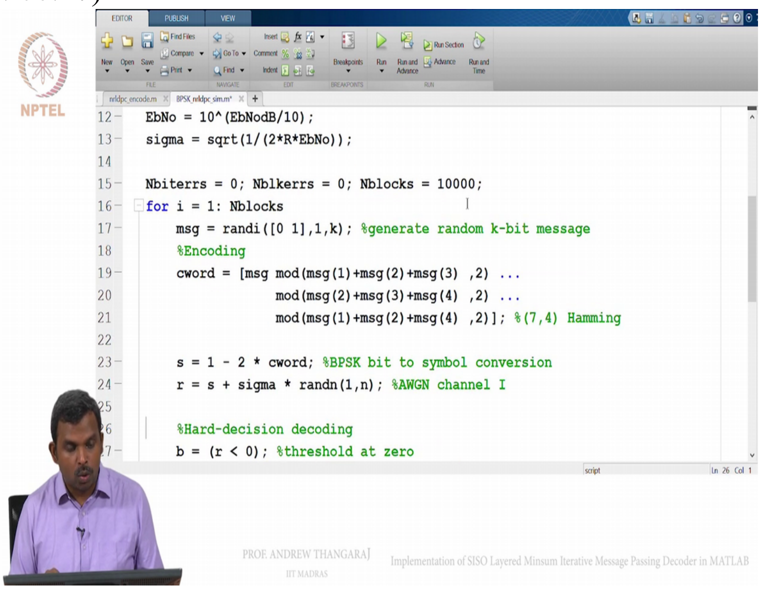

not needed, Ok.

(Refer Slide Time: 06:20)




So the other parts of the simulation remain the same. You generate the k-bit random sequence and then

(Refer Slide Time: 06:27)

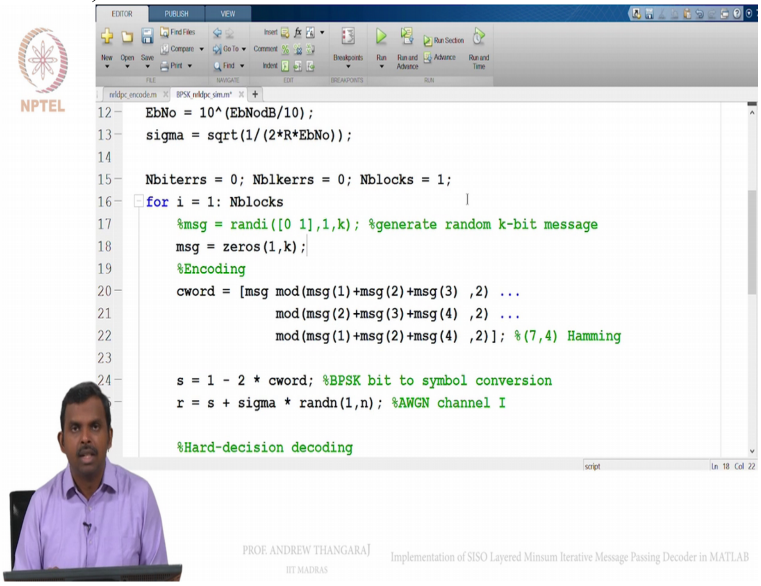



you start encoding.

So now remember I want to focus on the decoder. I do not want to be stuck with, stuck with; I am just testing my decoder in some sense. So I will not be doing a lot of blocks. I will just do one block and I do not really want to be encoding. So I will just comment this out.

I am not going to be encoding. And what I will do instead is I will simply assume the messages 0s. Ok so I

(Refer Slide Time: 06:55)

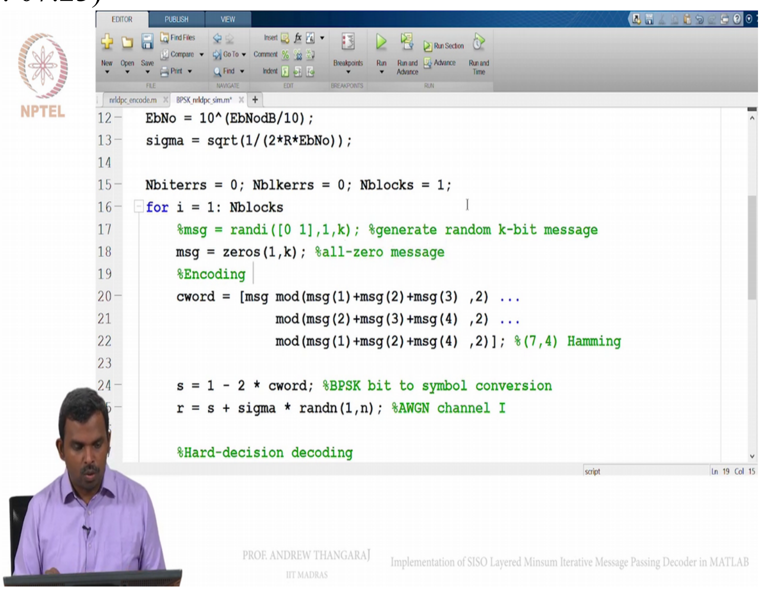



will transfer the all-zero codeword always, Ok all-zero message and all-zero codeword always.

It turns out that is enough for decoding checking of course and the decoder I will not assume all-zero codeword is being transmitted. I will try to decode it based on the received codeword.

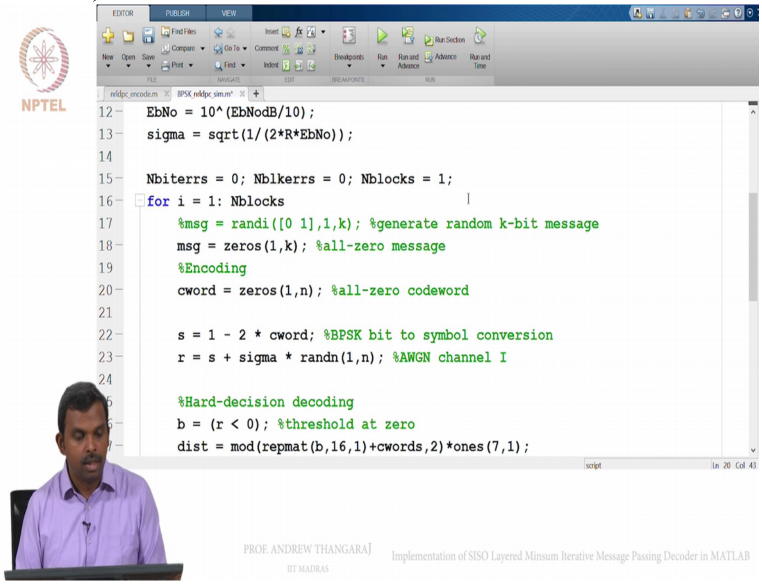
And this way I bypass the encoding requirement, Ok. We know we have an encoder input program wherein it just adds the complications. We can do that later on. For now onwards assume all-zero codeword, Ok.

(Refer Slide Time: 07:23)



So you know, all of us know when all-zero codeword is encoded you get, all-zero messages encoded with any linear code, you are going to get the all-zero codeword, Ok.

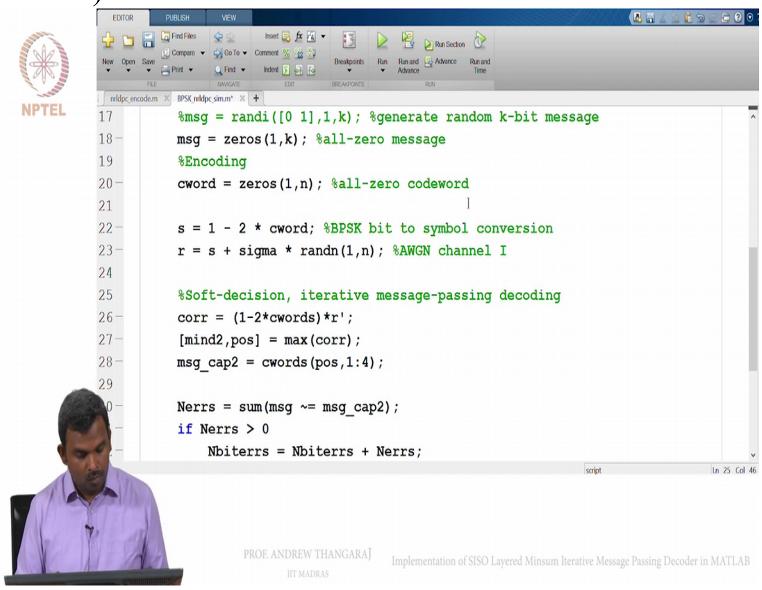
(Refer Slide Time: 07:40)




So this is something we do for simplicity in the decoder.

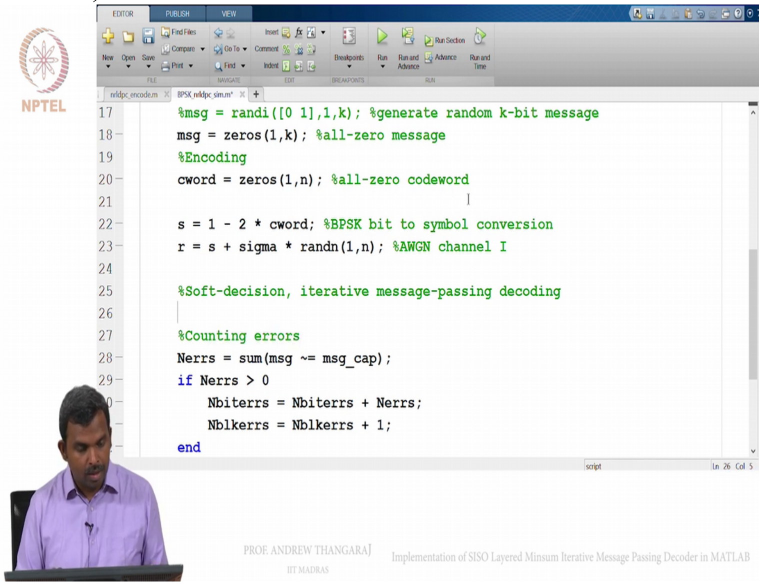
Ok so after that things are Ok. It is exactly the same, nothing much to change. We are not going to be doing hard decision decoding. We are going to be doing soft decision decoding. In fact this is going to be soft decision iterative message passing decoding.

(Refer Slide Time: 08:04)




So all of these things do not apply. You will have only one message cap. So this should be good enough. This is counting errors, right, so let me put that as counting errors.

(Refer Slide Time: 08:19)

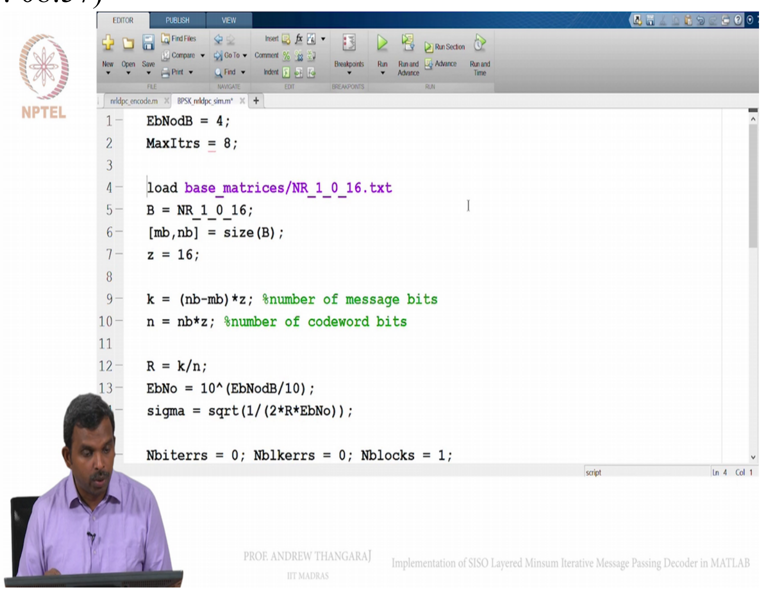



So now all that remains is the decoder. Ok so hopefully we have seen I have got the setting completely done, everything else is fine, I only need to figure out the decoder, Ok.

So I have loaded my base matrix, I have loaded my expansion factor, I have encoded and I have my received vector and I have to do my decoding. So all I to do is decoding, Ok. It is simple enough.

So in the decoder we know it is iterative so we need to figure number of iterations. So maybe that is the parameter you want to set. We could it here. Max iterations equals, I know 8, does not matter

(Refer Slide Time: 08:57)

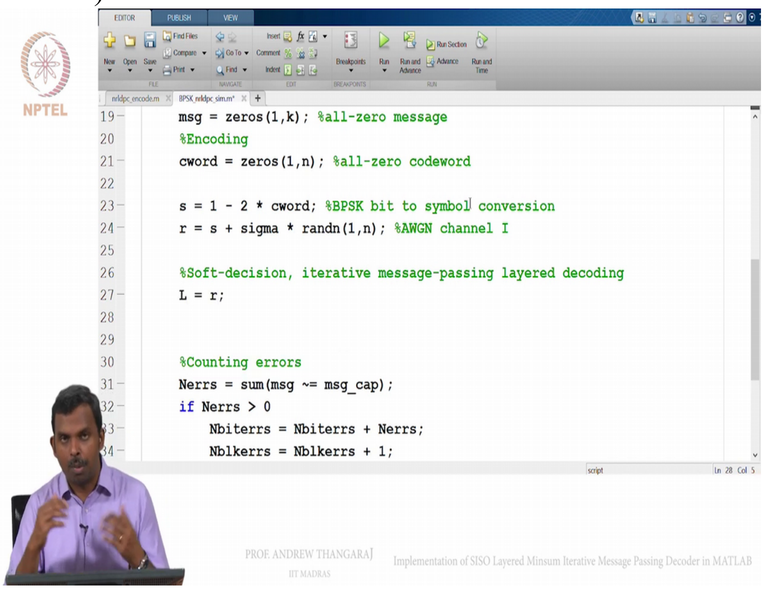


what we put here.

So, so that is the max iterations, Ok. We will keep that. We will use it, when we have to. So now the iterative message passing decoding, if you remember we are going to do layer decoding. So let me, layer I will add that, Ok.

If you remember layer decoding we have one total belief vector which is initialized to r in the beginning, Ok so that is my L. Ok

(Refer Slide Time: 09:26)




you can go back and look at the slides if you like to see where this notation is coming from. I did a toy example with L being the total belief, Ok so now that is, that is set as r which is the received vector. Now I am going to start working with r, right. So you remember I am going to process one layer at a time and for me one layer will be one block row in the base matrix.

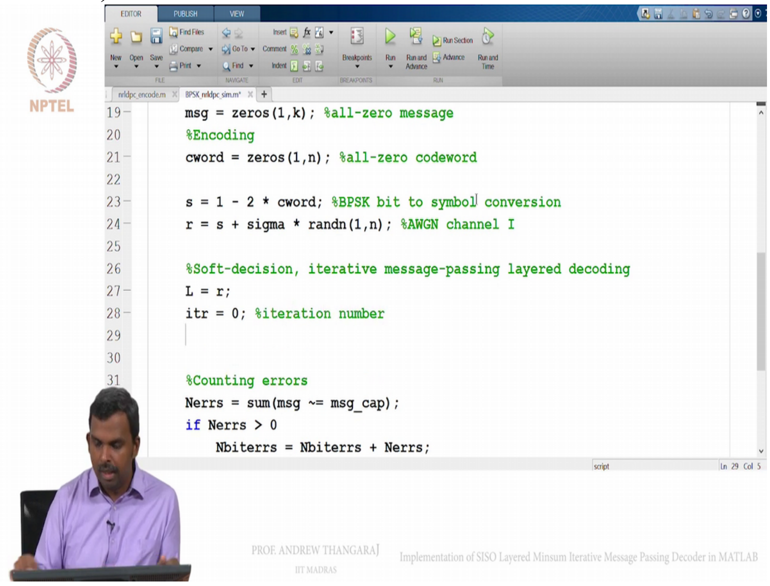
So you remember my base matrix has 46 rows. I am going to take the first row and process it. Ok. I am going to treat my first row as my first layer, Ok. So I will process that and then add to the total belief and proceed, Ok. So that is what I am going to do first, Ok.

So L equals r and then I have to start doing the first row. So how do I do the first row? So if you remember B is my base matrix, it has a bunch of minus 1s, Ok. And those minus 1s are not very relevant for me. I just have to go through and find out where the non minus 1 values are, Ok and then based on that do my row processing, right.

So that is, that is how it works. So it is, you can, you can do a lot of efficiency into this. Once again I am not going to worry too much about the efficiency. So the outer loop, the outer loop will be the number of iterations.

So I will have i t r be equal to 0. This is the iteration number, Ok.

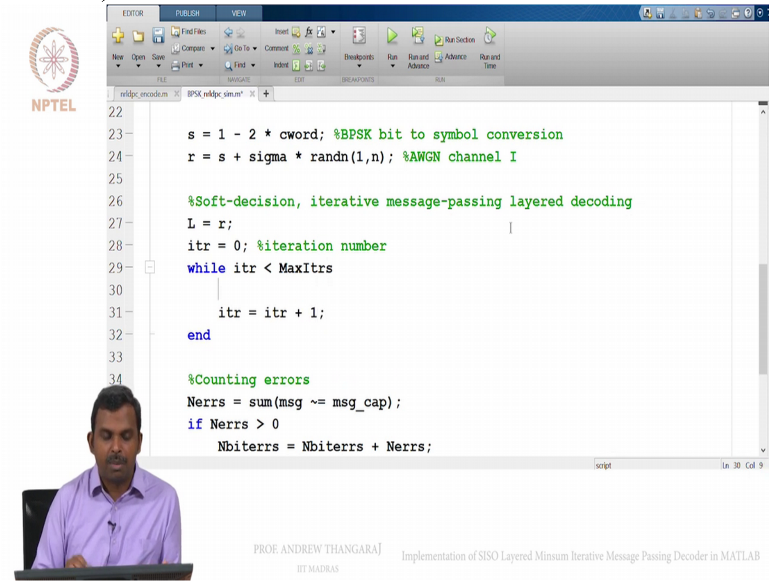
(Refer Slide Time: 10:59)




And I will also have, so there are various ways to do this thing, so, so one can put a loop here on while i t r as less than max iterations, right and end, Ok then over all I will have i t r equals i t r plus 1, Ok.

So you may

(Refer Slide Time: 11:25)

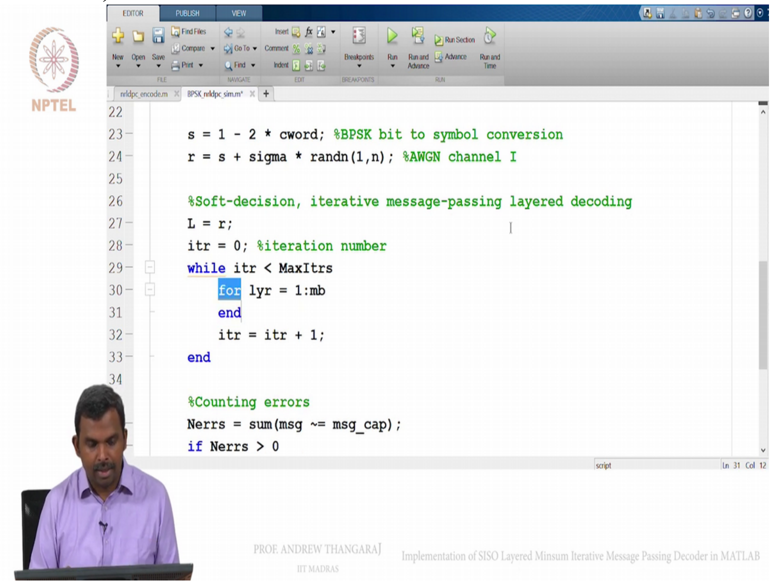



me why I am not doing a for loop. So it turns out I want to terminate in the middle if I have valid code word as the answer. So that is why I like a while loop here. So this is my loop for the iterations, Ok.

So I will repeat for every iteration a certain, the row processing steps, right, process all the rows. So I put the loop for the iterations, Ok. And inside this iteration I am going to process the parity check matrix and multiple layers here. And each layer I will need a loop as well, Ok.

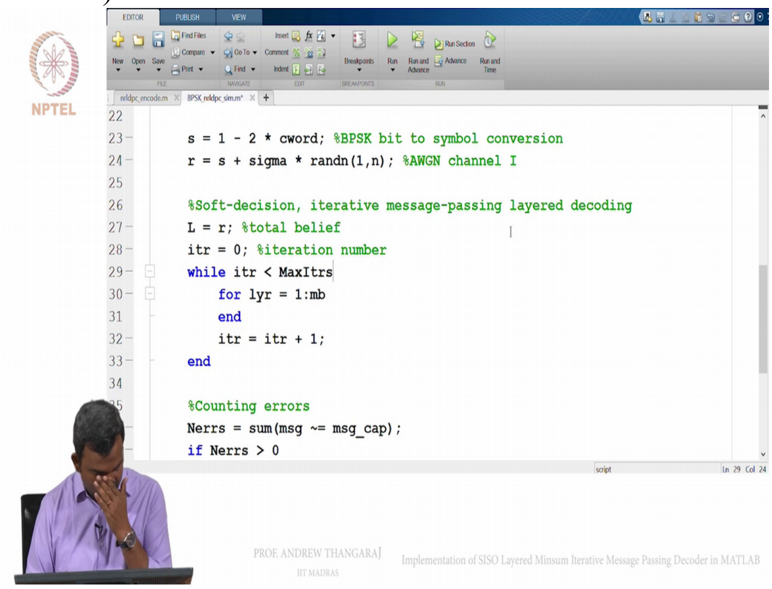
So that is going to be, I will call it layer equals 1 colon m b, right. So that is my number of layers

(Refer Slide Time: 12:05)




and that ends there, Ok. Once again what is L? L is my total belief, Ok.

(Refer Slide Time: 12:13)



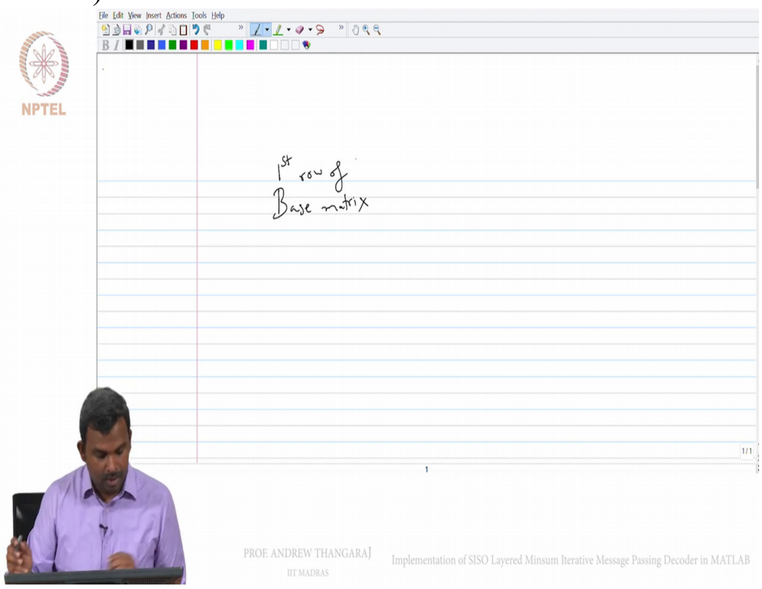

So lot of people call belief as log likelihood ratio, I am just using a slightly more generic terminology here.

If somebody says L is log likelihood ratio, technically it is proportional to log likelihood ratio. There is 2 by sigma square factor there. So that is why I am calling belief.

You can use L l r as well, I mean that is not wrong. That is quite reasonable to say, Ok. And what do we do in each layer? In each layer I have to do row processing, Ok. So it turns out, most people like to do this in parallel. So one can do that as well. And we will do some sort of parallel effort here.

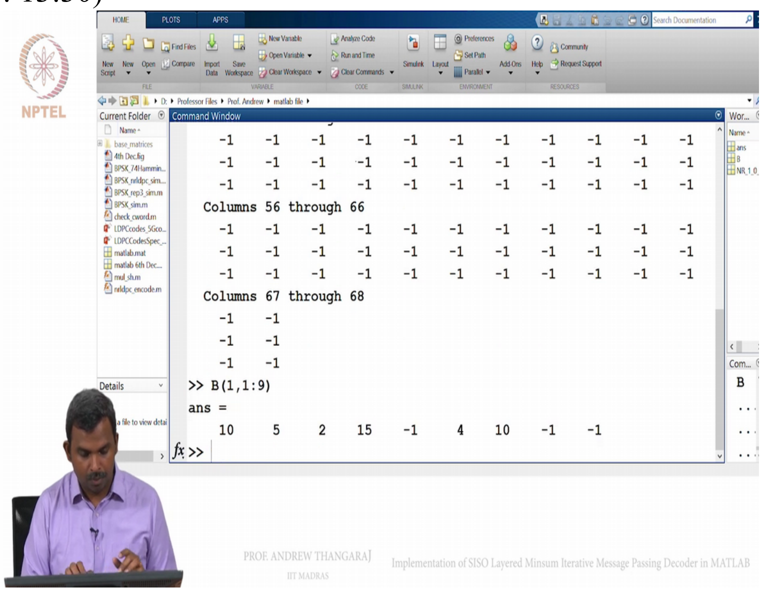
So, so remember what is going to be each layer? So maybe I should take out this thing? So if we look at each layer, so, so the base matrix B, base matrix, it is basically the first row of base matrix,

(Refer Slide Time: 13:21)




we saw that here (()) 13:25 B of 1 comma 1 colon 9

(Refer Slide Time: 13:30)

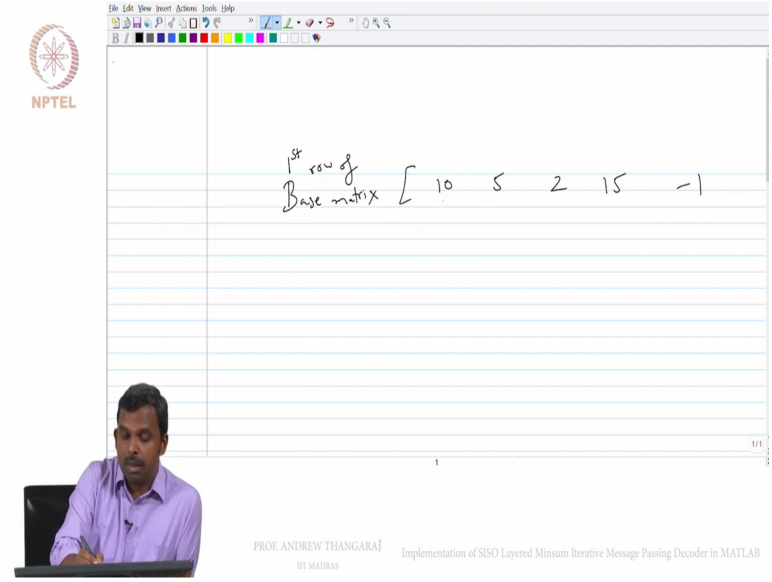



or something, so that comes in the first row.

So you see the first row is 10, 5, 2, 15, minus 1, 4, 10, minus 1, minus 1. There are lots of minus 1s but there is 10, 5 etc. So let me just copy this over to inside, so...I do not need to be exact. Just, just for rough purposes, this is 10, 5, 2, 15, minus 1 Ok. So you had 10, 5, 2, 15, minus 1.

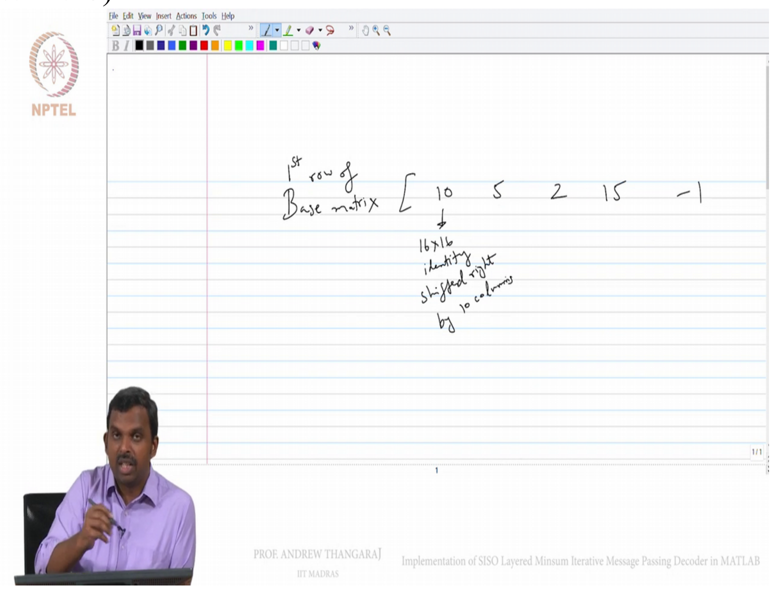
Remember what is 10? 10 is 16 by 16

(Refer Slide Time: 14:03)




identity shifted right by 10 columns right? So rotate shifted

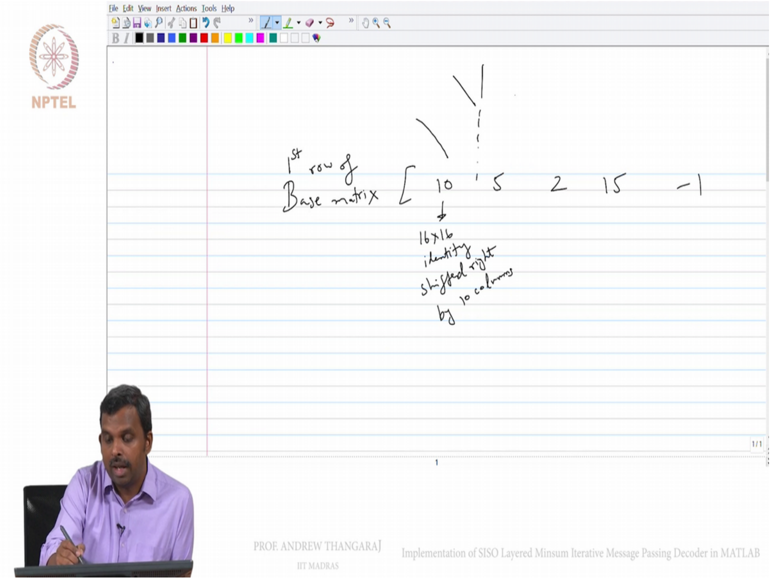
(Refer Slide Time: 14:17)




right by 10 columns. So it will have something like this, right.

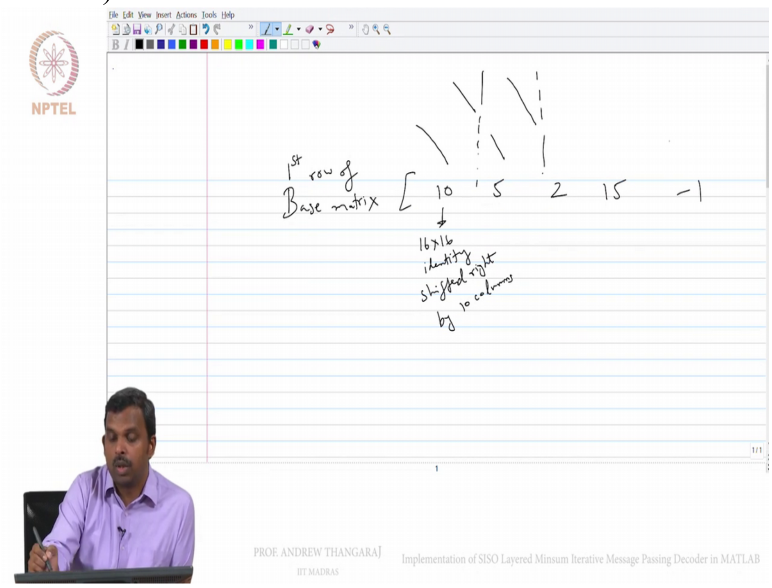
10, that would move here Ok something like that. So this would be this block and then the 5

(Refer Slide Time: 14:31)




would come. 5 would be probably longer here and then shorter here, Ok and then 2 etc,

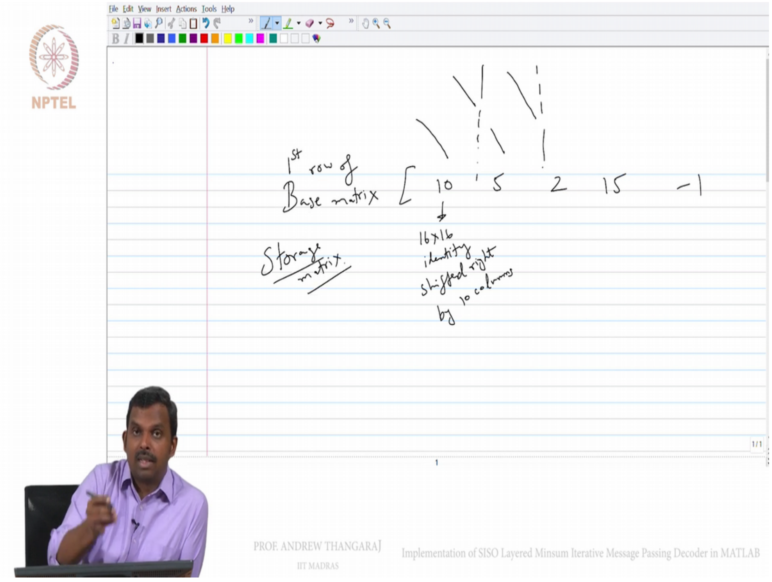
(Refer Slide Time: 14:38)




Ok. So the parity check matrix is expanded from here. And there is some shift going on here, Ok. That is number 1.

Number 2, if you remember every entry in the base matrix I need a storage matrix,

(Refer Slide Time: 14:57)




right. So that is the message that is being exchanged. So that storage matrix actually needs for the expanded version, right? So I need a fairly big storage matrix to work with. Ok without the storage matrix things are not going to work.

Ok so that is the next big thing that we will define carefully, Ok. So how will we define the storage matrix? It needs to go outside of, outside of this you define a storage matrix based on B, Ok. And how is the storage matrix going to work?

And remember I do not need a storage for every entry in B. B has a lot of entries, 46 by 68 but quite a few of them are minus 1s, Ok.

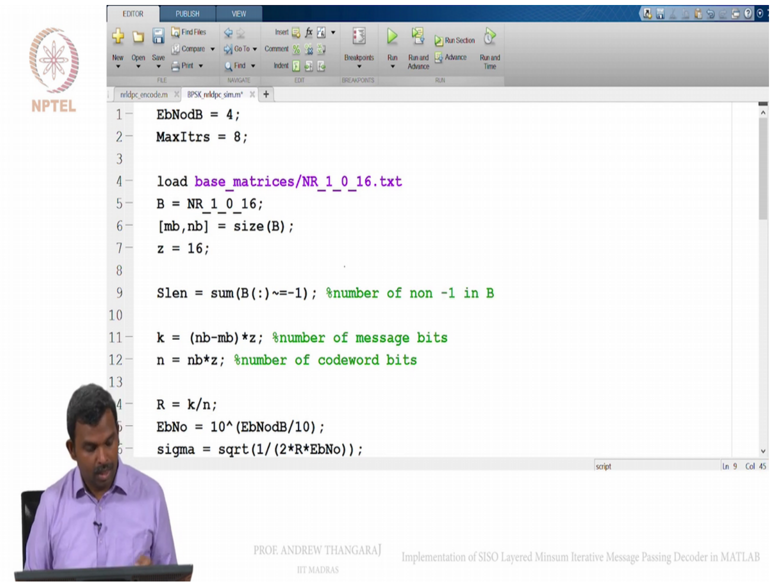
For every minus 1, I do not need to store anything. Only if it is not minus 1, I need to store something. And how much do I need to store? For every entry, non-minus 1 entry of B, I need to store 16 values, right. That is the expansion by 16, right. That is only identity, Ok.

So 16 1s are there for every entry in the base matrix or more generally, z 1s are there for every entry in the base matrix, Ok. So I will find out how many non-minus 1 entries are there, where they are and, and sort of where they are in this will give me and wherever I have a non-minus 1, I need to store 16 values, Ok.

So I will store 16 values in a sequence and then we will have to do some rotations. We will come to that. We will deal with this but for now let us declare this storage array. Ok that is quite important. We need to do that, Ok.

So how do we do this? So, so first I would need to find number of non-minus 1s. So I will call it S len equals length of, or well you can even do this, sum of B of colon not equal to minus 1, Ok. So this is going to give you a total count of number of non-minus 1 in B, Ok.

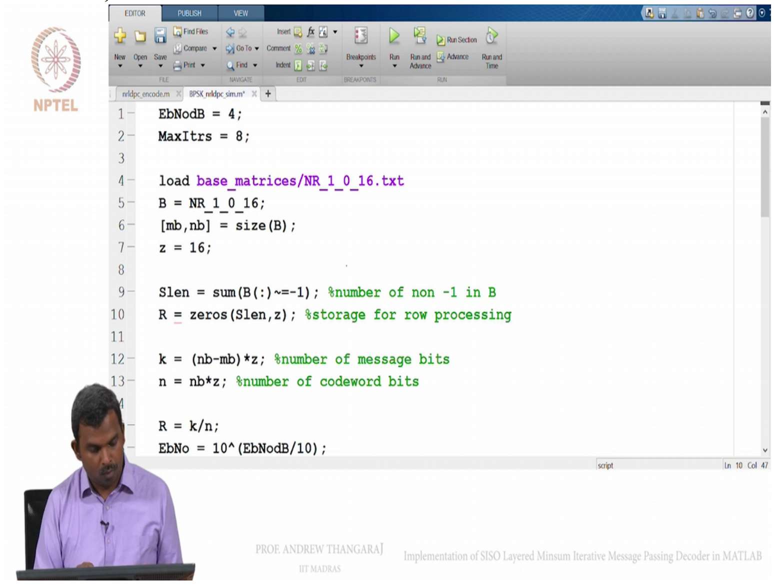
(Refer Slide Time: 17:08)




Ok.

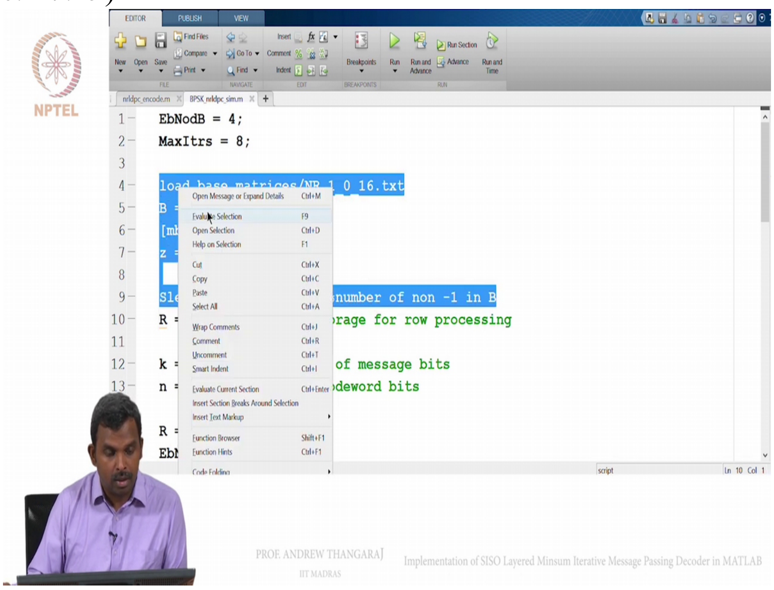
So why did I do colon? I just wanted to make into one vector, so that I can do this quite reasonably. So you can, you can get back and then once you have S len, you can define your storage, Ok. So my storage, I am going to call it R, R is zeros for now of S len comma z, this is the storage for row processing.

(Refer Slide Time: 17:41)



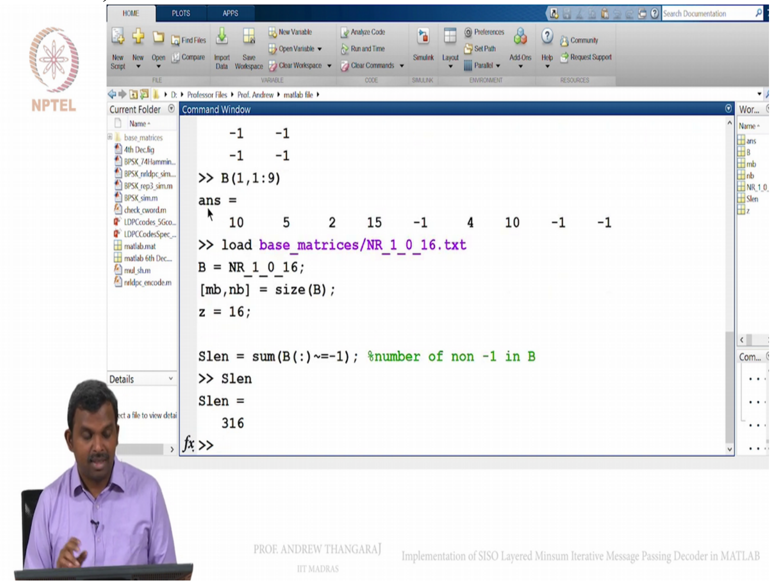

Ok, so hopefully you agree that this is correct. So maybe we can run up to this and see what the value is,

(Refer Slide Time: 17:49)




evaluate selection. If you do that you will see that S len is. It is

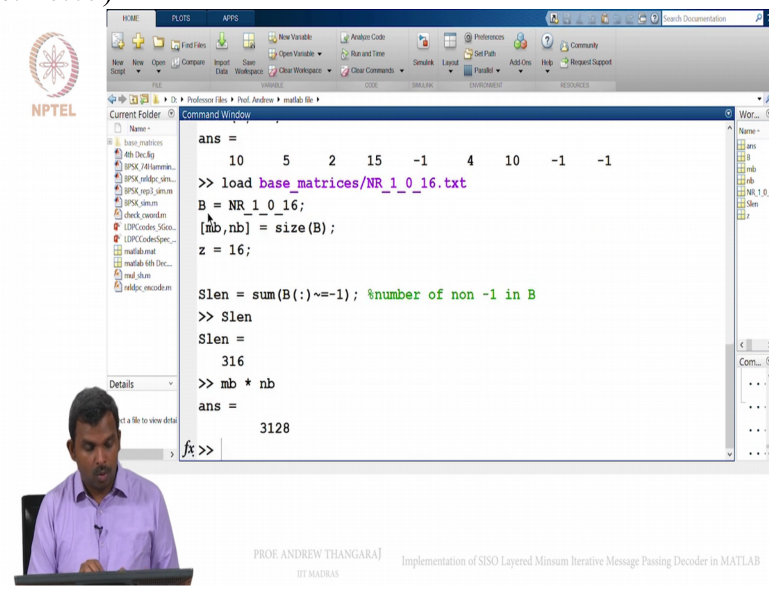
(Refer Slide Time: 17:57)




316, Ok.

So even though there are, so if you remember, B is, if you do m b times n b

(Refer Slide Time: 18:05)

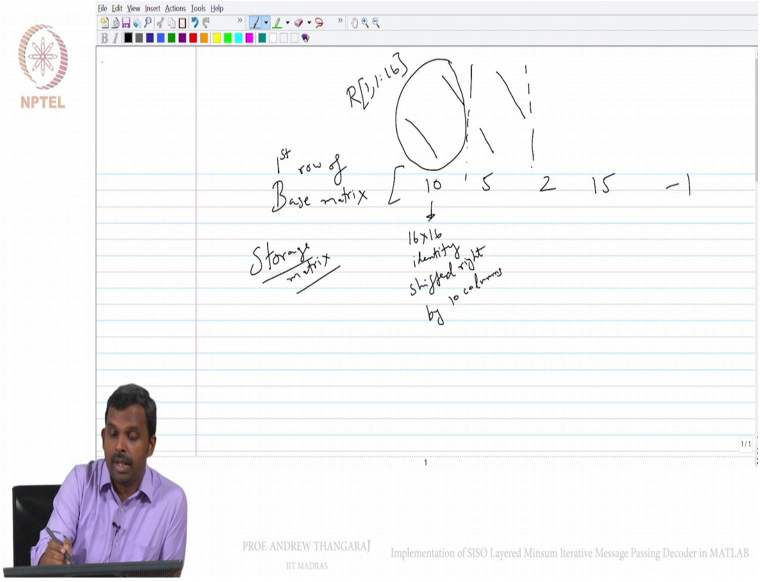



you get 3128, so that is the size of, total number of entries in B is 3128, 46 by 68 but out of that only 316 are not minus 1, Ok.

So we need memory only of that length, 316 times z. Ok what is that z? It gets expanded by z and that many 1s are there in the final matrix, Ok. Or so that is good. So we have done that. We have this R, Ok. Now this R is sort of like a linear entry so the every non minus 1, I have R Ok, so let me show you why it is important.

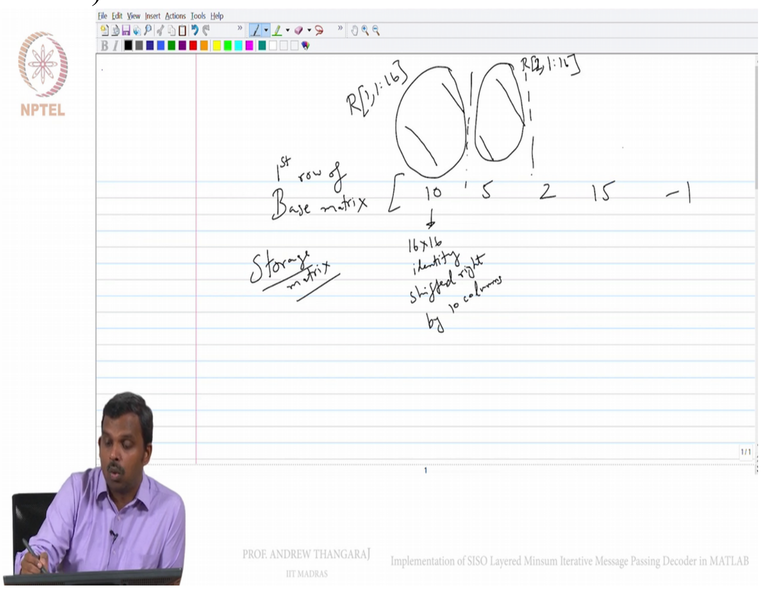
So the storage for this, Ok storage for this will be R of 1 comma 1 colon 16, right.

(Refer Slide Time: 18:52)



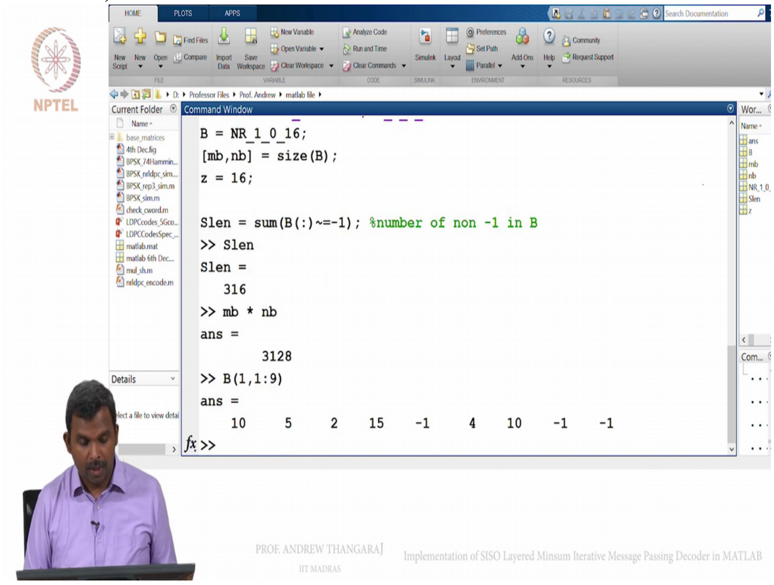

What will be the storage for this? That will be R of 1, 2 comma 1 colon 16 and

(Refer Slide Time: 19:00)




so on, Ok. And this will be R of 3, R of 4 and then there is a minus 1 and if you remember the matrix B, Ok,

(Refer Slide Time: 19:13)

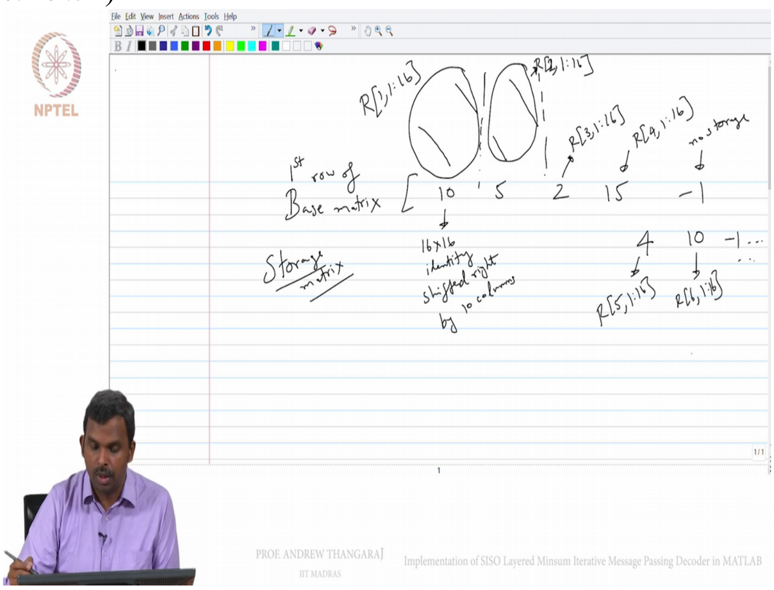



there was the minus 1 and then 4, 10, minus 1, minus 1, Ok. Let me go to that once again.

After minus 1, you have 4, 10, minus 1 so on. So this will be R of 2 16, the storage corresponding this will be R of 3 comma 1 colon 16, the storage corresponding to this will be R of 4 comma 1 colon 16. There is no storage here corresponding to minus 1. And the storage corresponding to this is R of 5 comma 1 colon 16, and so on.

This will be R of 6 comma 1 colon 16 and so on, right. And so it will proceed like this. So, Ok so it should be,

(Refer Slide Time: 19:54)



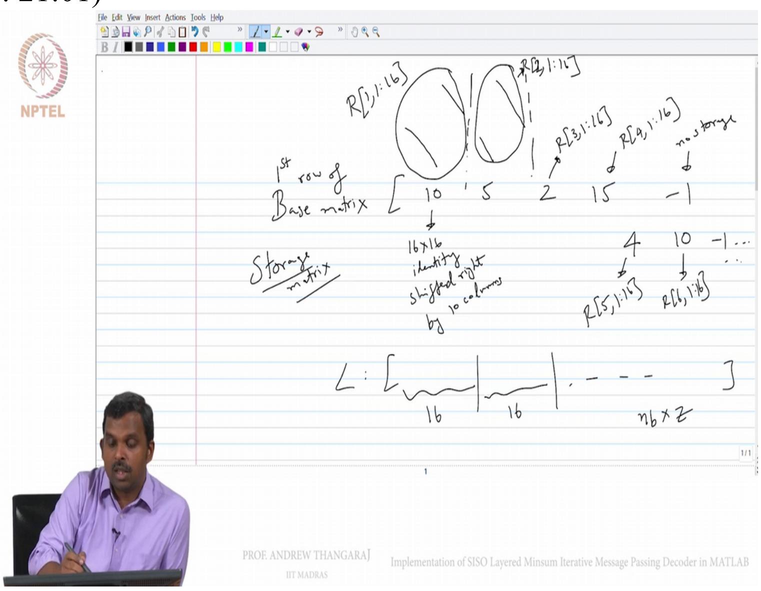
so, so hopefully that is clear to you. So this is how I am going to organize my storage, Ok. So it goes row wise, Ok and wherever there is a minus 1, I will just skip that index. I will go to the next index in this. Is that Ok? Hopefully that is clear, Ok.

So once again remember my storage is just linear. I am storing z values for each non zero non minus 1 entry in the base matrix and, it just goes linear. As you traverse row wise, I will go next memory, next memory, next memory like, Ok, keep that in mind.

So that is number 1. So as I process the first layer like this, I know, I know what to do. So I come here, I have my storage. So I have my access to R. I have the L, the total belief corresponding to this. Remember the L will come out, L is there, right. So that is just for the whole n b into z.

Ok so you will have the first 16, then the next 16 so on, Ok. So you will have the first 16 Ls corresponding to

(Refer Slide Time: 21:01)

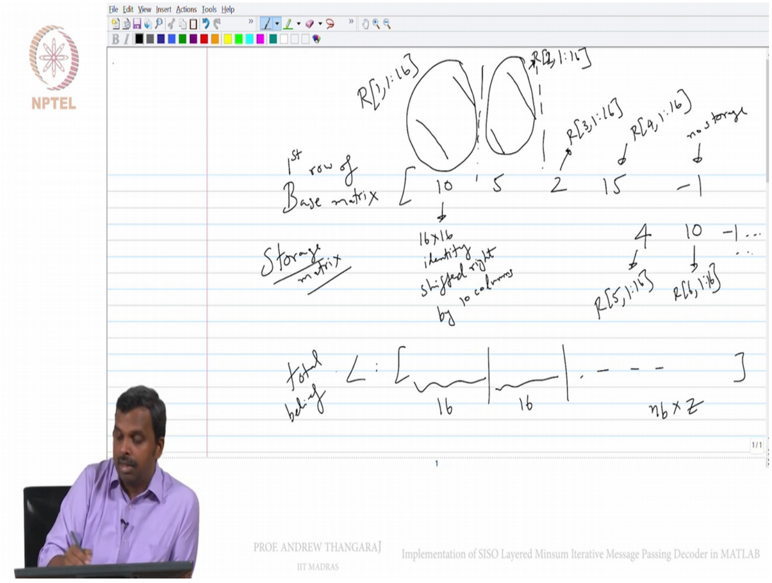



this, the R will be 1 comma, 1 colon 16. But remember there is this, little bit of shifted thing there, right. So one needs to pay attention to that. When you align it with the row but as far as columns are concerned, it is the same, Ok.

So I am going to subtract. You remember the row processing. I do L minus R first, Ok. That is the subtract operation, layer decoding. And then you do the row processing, Ok. There you have to align with the rows and then do the row processing. And then the L minus R and the new row get added to get the new total l l r. Ok so that is the operation that we have to do.

So we will do all of that. So one needs to exercise some care when we do this. Nevertheless it is, hopefully it is easy enough to remember, Ok. Alright so that is this picture here.
So hopefully, it is clear to you. You have the total belief L, Ok so you have the total belief here, L which is

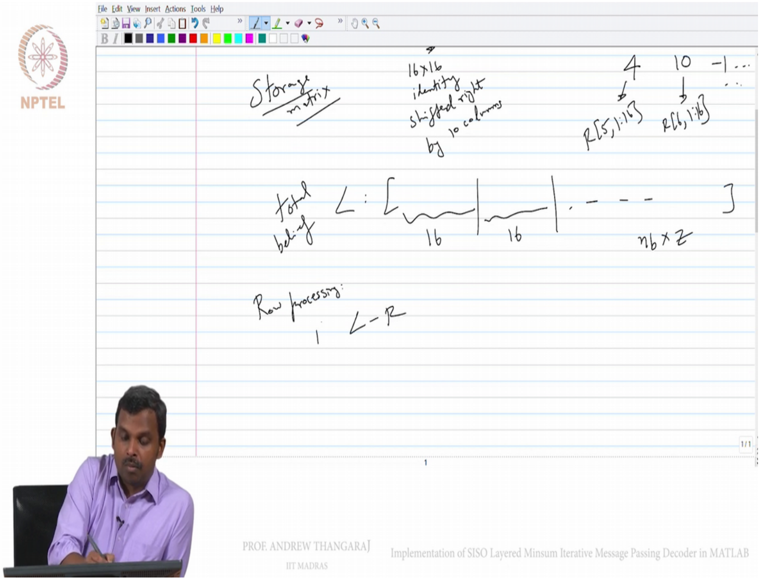
(Refer Slide Time: 22:00)




in one vector and then you have this R matrix which is storing sequentially as you go along the rows, what your R values, initially it is all zero but still you should store that and then you do, then you have to do row processing, Ok.

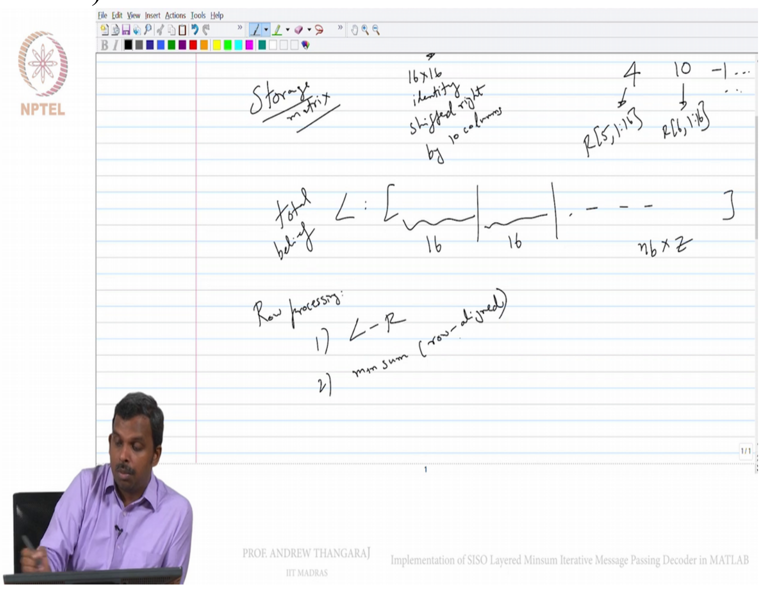
Row processing, row processing involves L minus R. That is the first step.

(Refer Slide Time: 22:24)



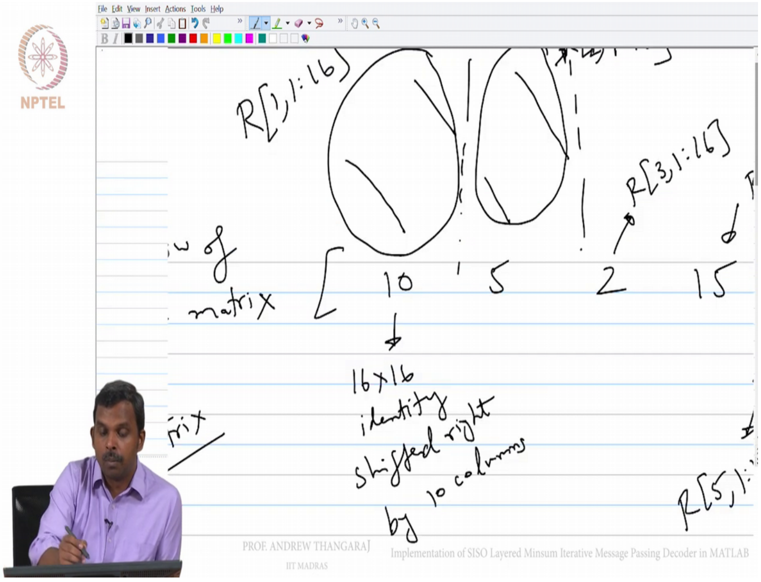

Next minsum, row aligned, Ok so you have to do a row alignment here.

(Refer Slide Time: 22:32)




Remember, why do I have to do a row alignment? Hopefully it is clear to you because,

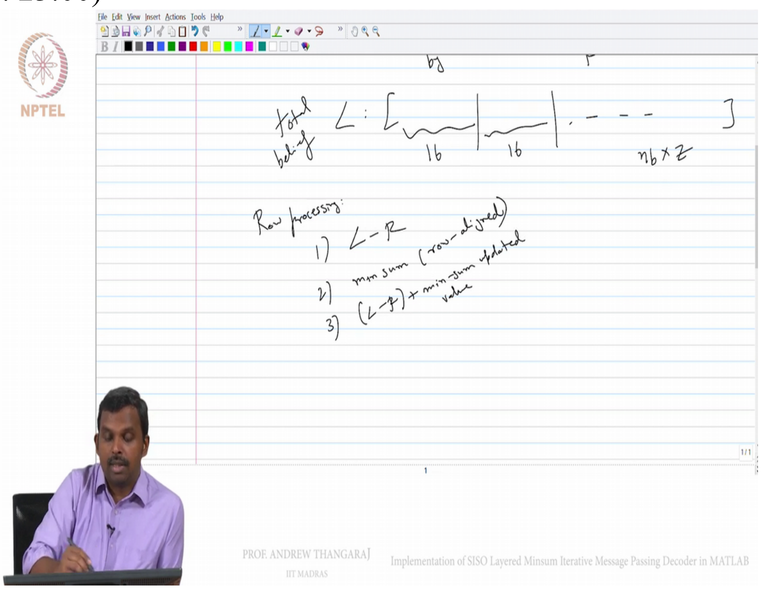
(Refer Slide Time: 22:35)




so while R starts like this; that is not aligned in the same row, Ok. So the first thing is what is aligned.

So you have to shift by 10 and then only you will get alignment in the row. Ok, those are the corresponding things that you have to work with. Ok and final thing is L minus R plus the, minsum value, Ok minsum updated value, Ok.

(Refer Slide Time: 23:00)




This is what I have to implement. And there is this row alignment which I have to be careful about, Ok. So to actually do the row processing, I like to sometimes pull in the values from the storage into some temporary storage that I will have and then work with that. Ok, so there are various ways to do that and one, so, so this also makes sense.

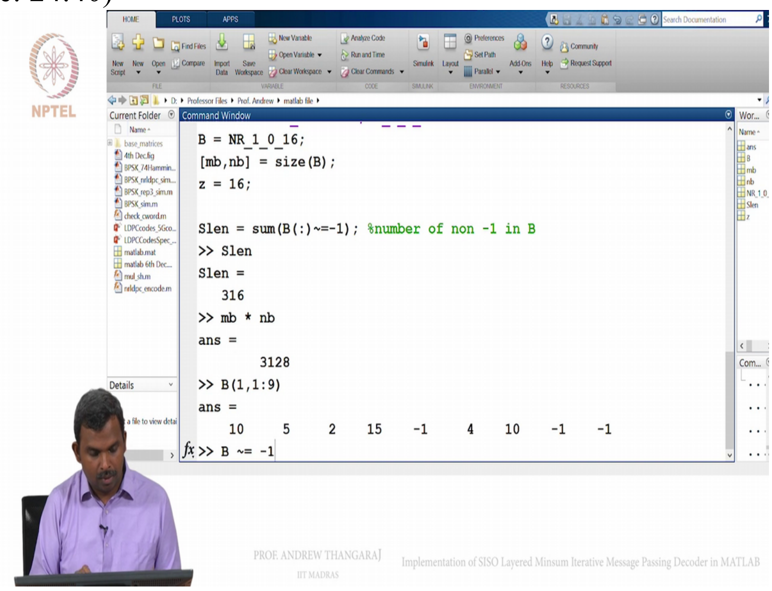
Because usually the storages is in some, when you actually implement in, let us say hardware, the storage is in some memory and you want to pull it in some registers, into a small operating unit and do the calculation and then write back into memory.

So this is something that you want to do to make sure things work cleanly. So I am going to use some registers, temporary storage for pulling in these, these R values. L is Ok; L is just available to me all the time. R values I am going to pull from memory into my sort of some temporary storage, do the processing and then store it back, Ok. So that will be my flow, Ok.

So let us see how to get this done. Ok, not getting the right thing, Ok there you go. So I need to define that also, Ok. So how many do I need? So for that I need to know the maximum number of 1s in a row, Ok. In the expanded base matrix, number of, maximum number of non minus 1s in the rows of B, Ok. So for that one can write small piece of code.

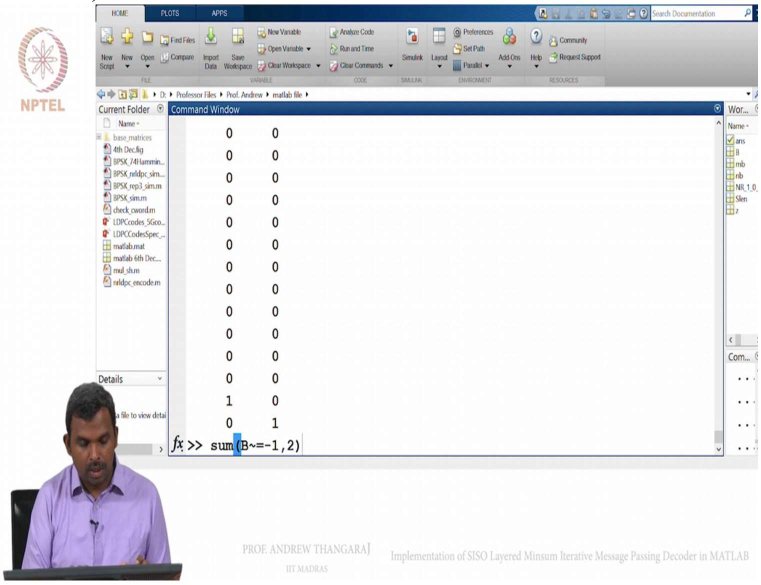
So if you look at B not equal to minus 1,

(Refer Slide Time: 24:40)



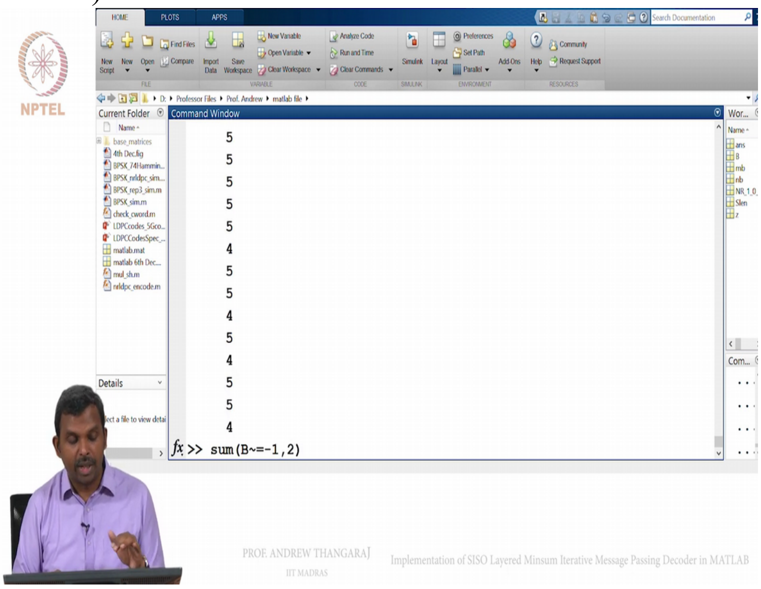

this will give me a matrix, Ok it is tough to see that. So if you add up B not equal to minus 1 comma 2

(Refer Slide Time: 24:50)



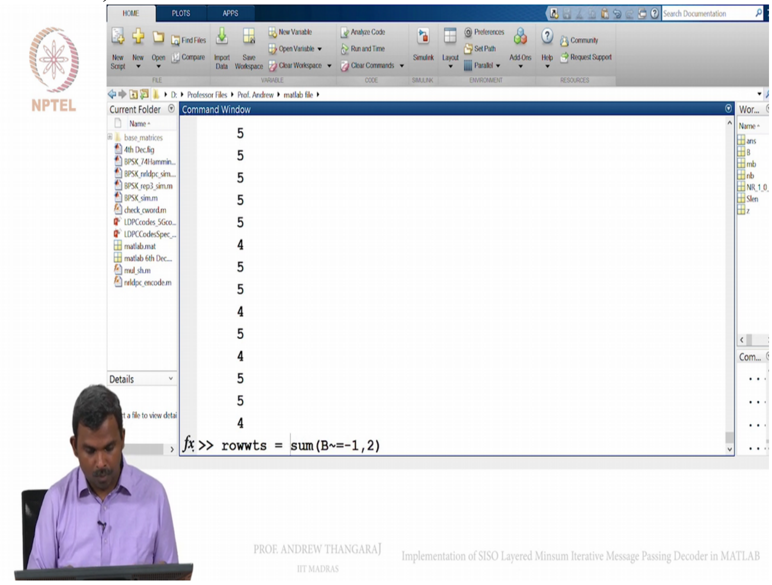

Ok

(Refer Slide Time: 24:50)



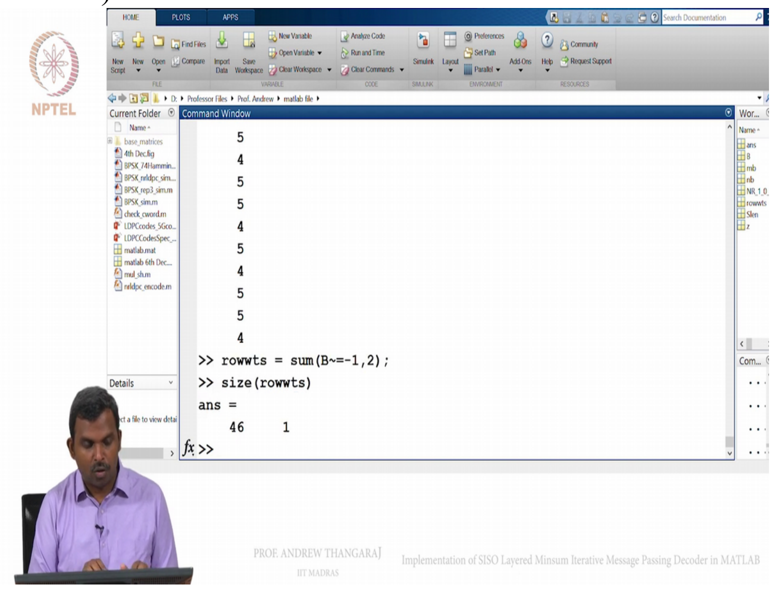

so this will give me, this will give me the row weights. Ok so if you want, I can define this as row weights.

(Refer Slide Time: 25:02)




Ok if you did not believe me, just look at the size of row weights,

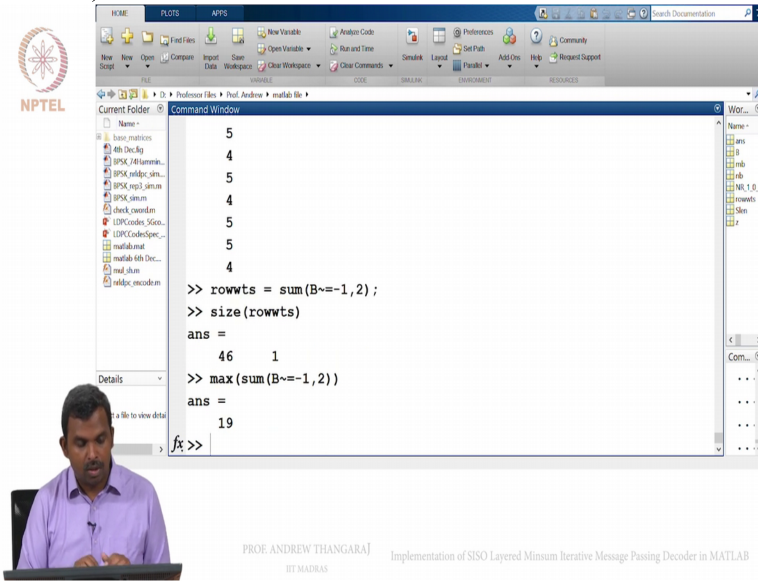
(Refer Slide Time: 25:11)




there will be 46 comma 1. So there are 46 rows.

So in fact I do not need all the, entire row weights. I can simply do max of sum of B not equal to minus 1 comma 2,

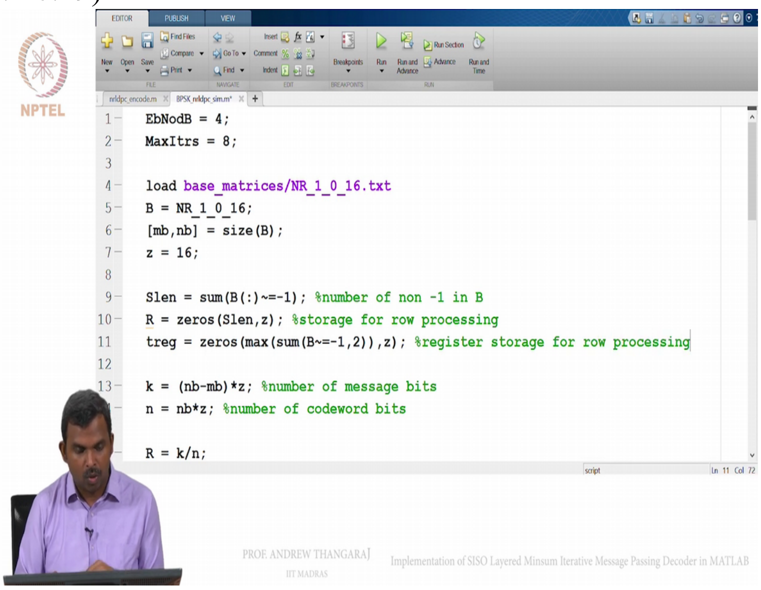
(Refer Slide Time: 25:23)




Ok. So there are 19, that is the maximum weight that you have in this matrix, Ok. So that is the number there. So one can do that also.

So, so, so, so let me define the temporary storage. I will call it t reg, temporary registers, Ok, zeroes of max of B not equal to sum of B not equal to minus 1 comma 2. I do not think I need this number anywhere else so that is why I am not worrying too much about it. That is it, Ok. So this is the, the storage, register storage I will call it for row processing.

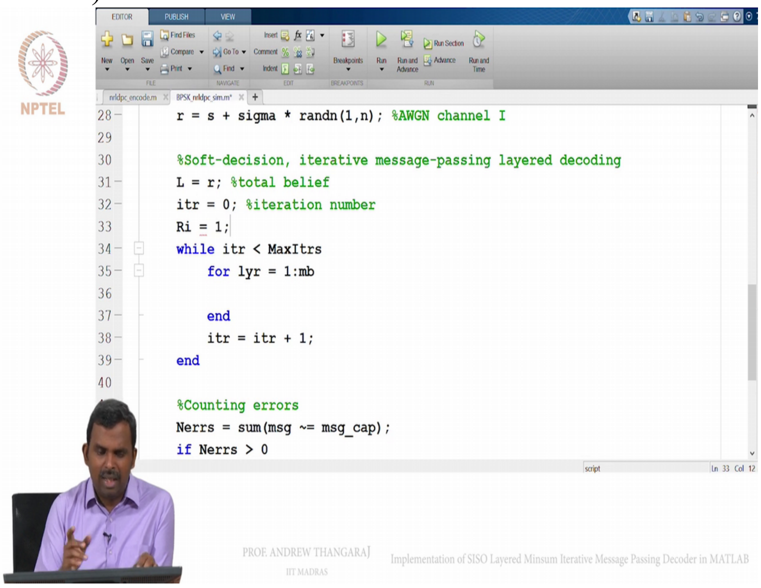
(Refer Slide Time: 26:19)




So all the row processing I will use t reg for, Ok. So let us do that.

Ok so now I come to the layer. So I have to now read the, the storage, the R storage into my t reg, Ok. So I also, it also depends on what layer I am in, Ok. So there are various ways to, to do this. So inside a layer, I am inside each layer, so I will have to keep a count on where I am in R. So I will, I will keep that also as R count equals, I will call it as simpler notation R i equals, we will have

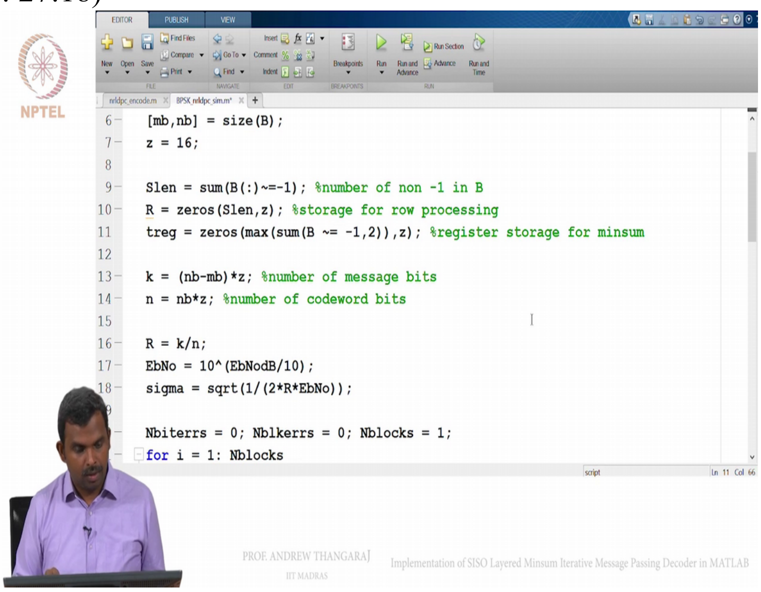
(Refer Slide Time: 27:10)




this one, does not matter.

So this is register storage for minsum.

(Refer Slide Time: 27:18)

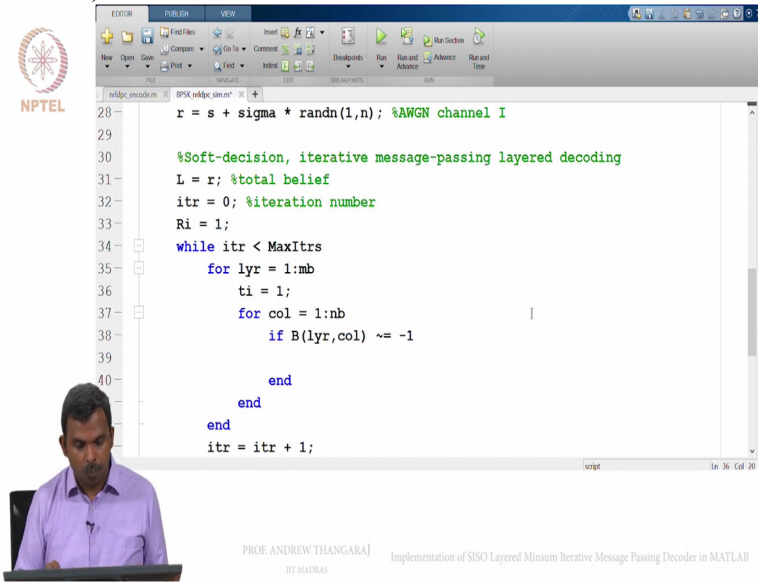



That is good to have. So let us go through. So I am going to go through here for each layer and for each column in the base matrix. I am checking if it is minus 1 or not. If it is not minus

1, I can store it into my register. But I will store it after doing subtraction and, and taking care of the index also.

So let me do this bit more carefully. So I will call it t i equals 1.

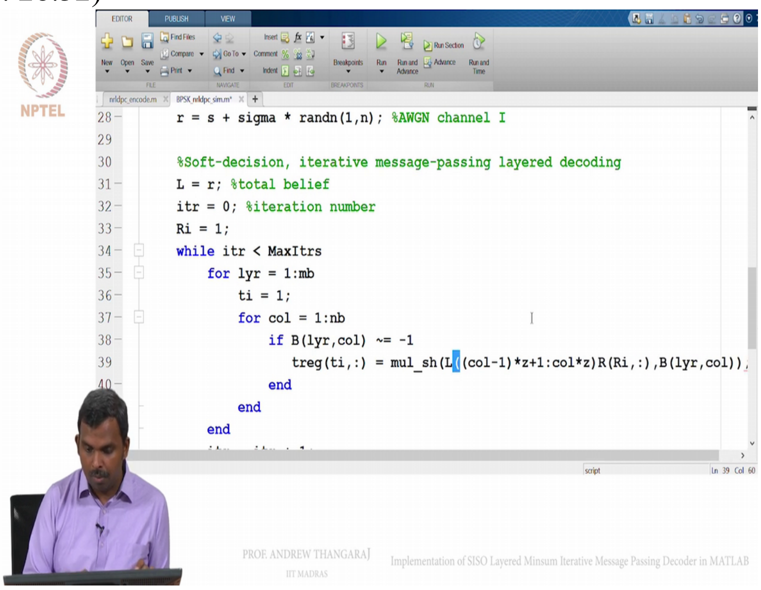
(Refer Slide Time: 27:47)




And then if it is not minus 1, my t reg of t i comma colon equals, I need to do a mul shift of R of R i comma colon comma B of layer comma col, Ok.

So I am shifting it and store. What should I shift? Not R, I need to do L of, the column is 1 so it needs to be col minus 1 multiplied by z plus 1 colon col into z

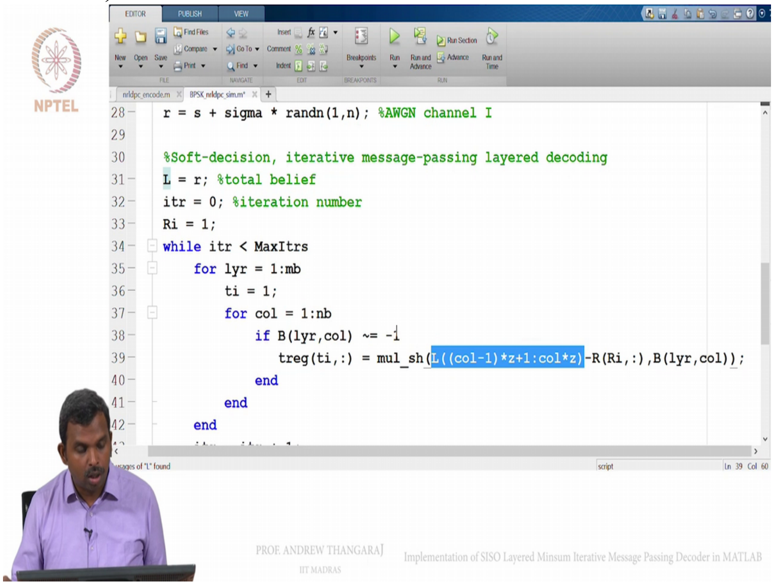
(Refer Slide Time: 28:32)



minus R i and then comma this, Ok. So let me just go through this. I just typed out a lot of things.

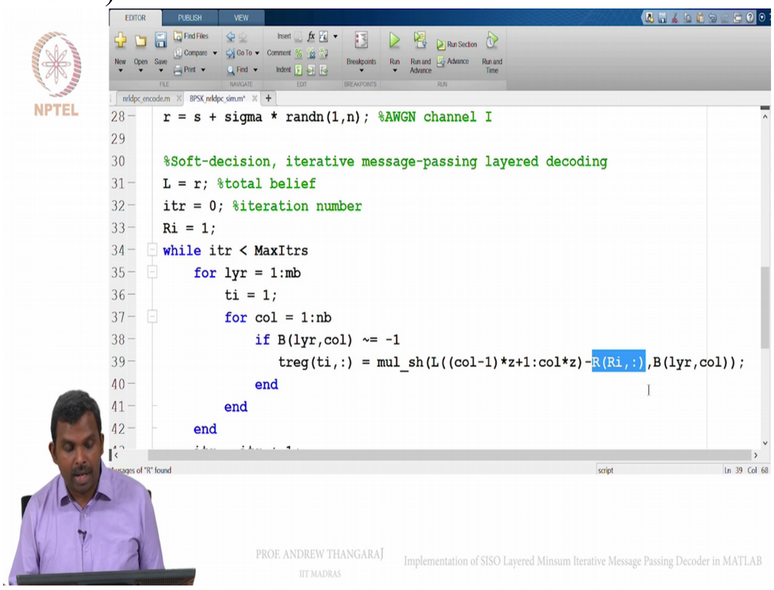
So you can see what I am doing here. I am taking the corresponding L. From L if I might,

(Refer Slide Time: 28:45)




a column c o l, then L of column minus 1 into z plus 1 colon column into z is the corresponding block of total belief from the entire L vector. And what is my R? That is R of R i

(Refer Slide Time: 29:01)

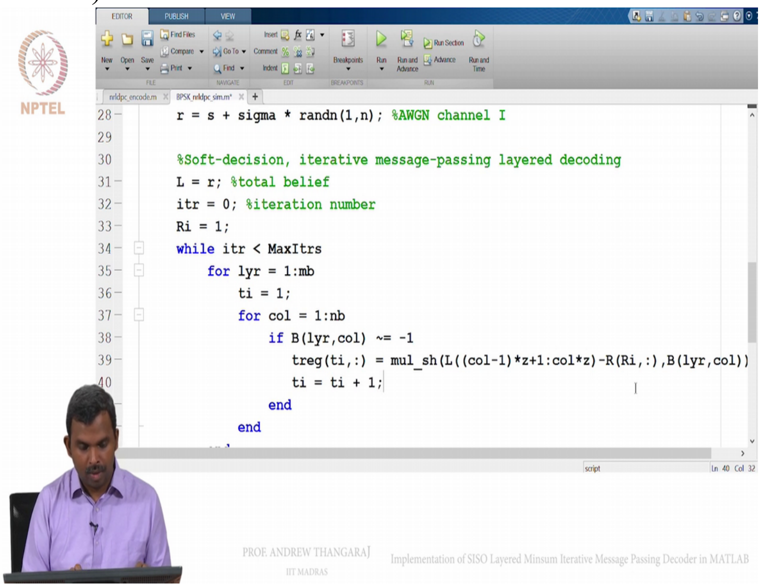



comma 1.

So R i is my index in the, in the message, the storage register for the row processing and there I take the R ith one but after I subtract I need to shift it. The shifting is to row align, Ok so that is the mul shift that we had. Even in encoder we saw this operation. I am going to shift it by the entry in the B, Ok. So that is one step lets me do this.

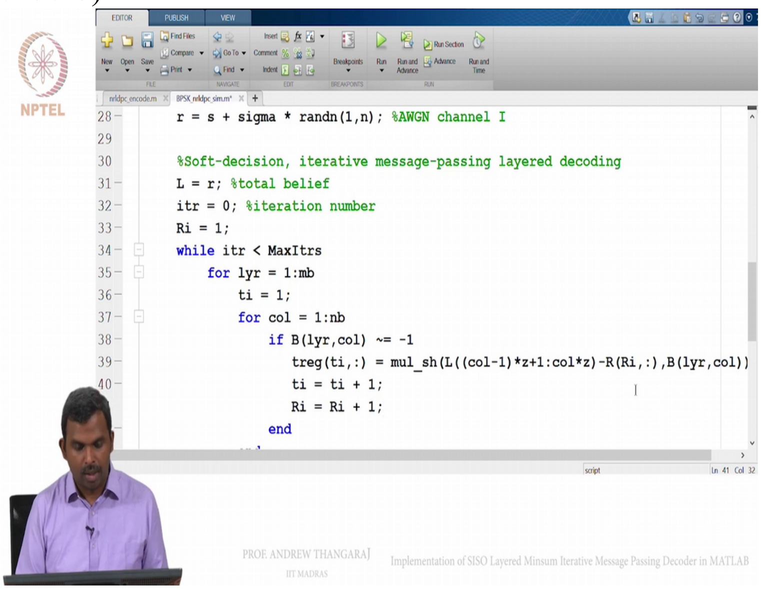
But then after I do this, I have to increment my t i

(Refer Slide Time: 29:33)




and increment my R i.

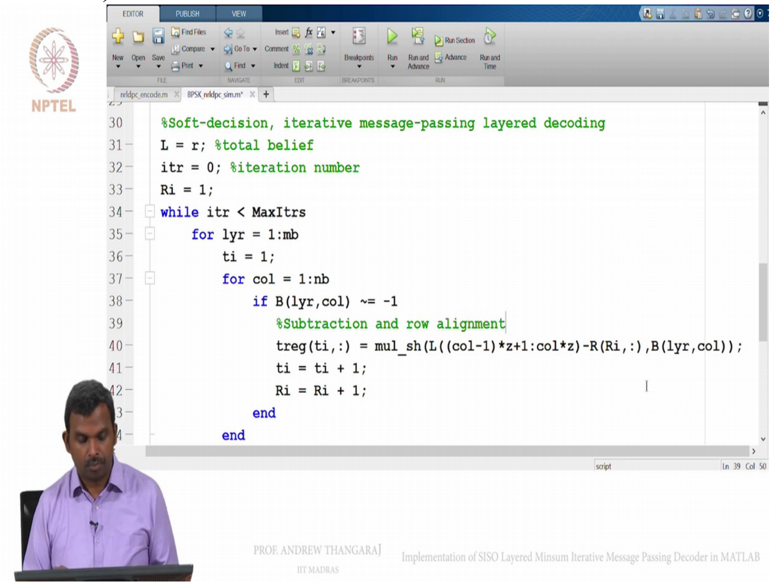
(Refer Slide Time: 29:37)




Is it Ok?

So once I did this, I have got all the values from the storage for the row processing into my register for minsum, Ok. Hopefully I will go through this once again. For every layer, I am look at every block column, I am seeing if it is minus 1 or not. If it is not minus 1, I have to process it. What do I do? This step actually does subtraction and row alignment.

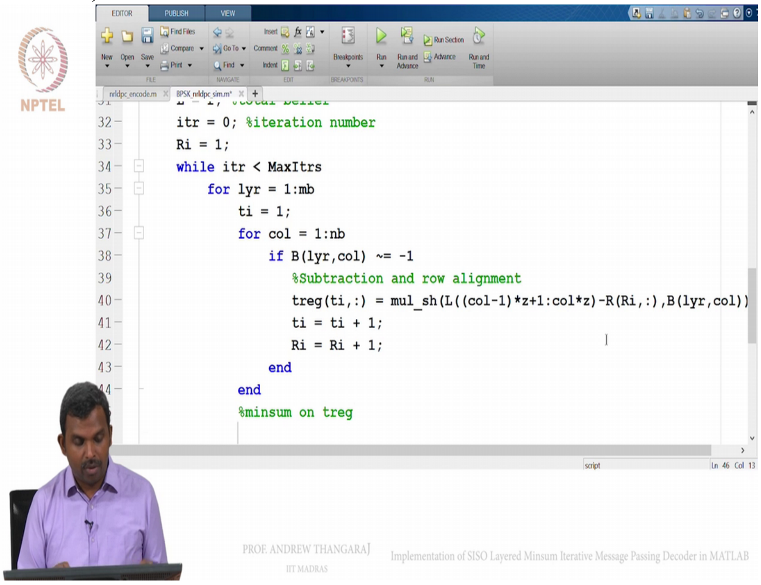
(Refer Slide Time: 30:09)




Ok, so it brings in the corresponding value from the row processing, row storage R and the corresponding value from the total belief. It subtracts the, subtracts the two and then shifts it by the actual value in the shift, in the, in the base matrix and then stores in t reg.

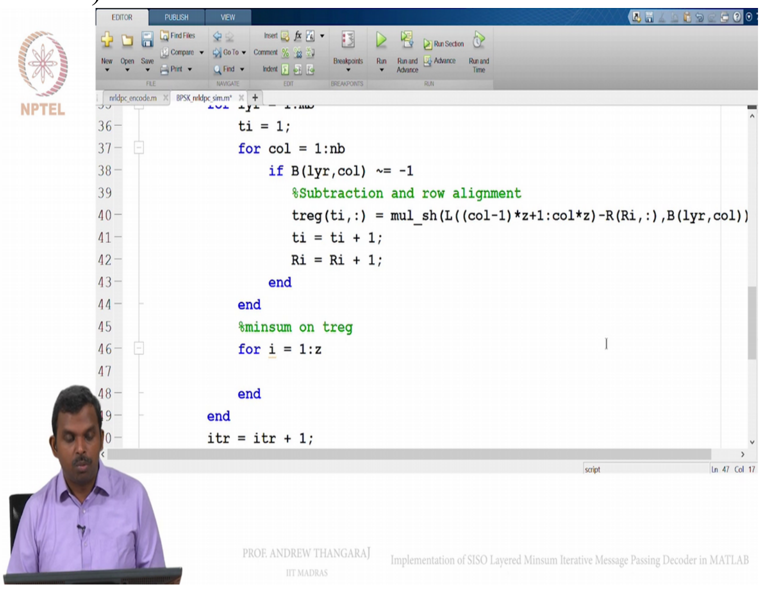
So at the end of this loop, once I am done with the every column here, I have all the values read, brought into, brought into this t reg register and they are all nicely row aligned, Ok. So now I am ready to do minsum, Ok, minsum on t reg, Ok.

(Refer Slide Time: 30:48)




On every entry in t reg I have to do minsum and that is, I will just call it for i equals 1 colon z, Ok

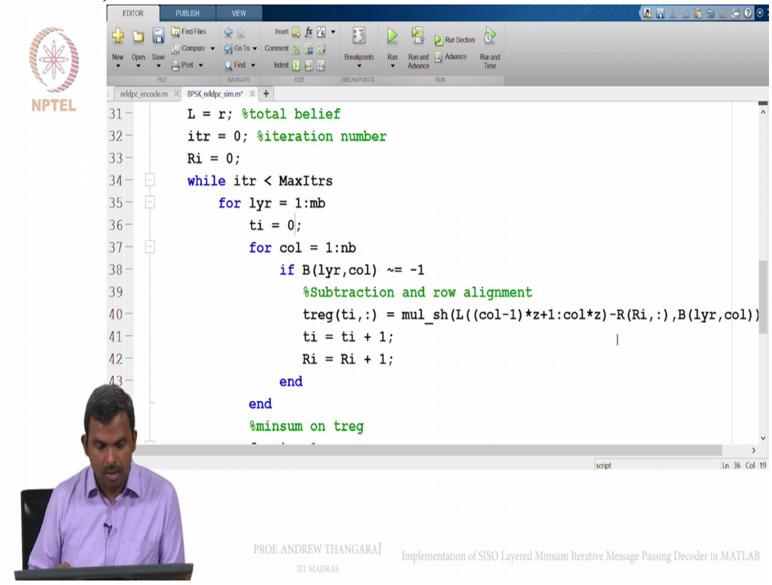
(Refer Slide Time: 30:58)




I have to do minsum, Ok. So, so doing minsum, you can do in various ways. I am doing a simple loop here. You can do the without loops also if you like. I am going to do it just for simplicity; I am going to do it with loops.

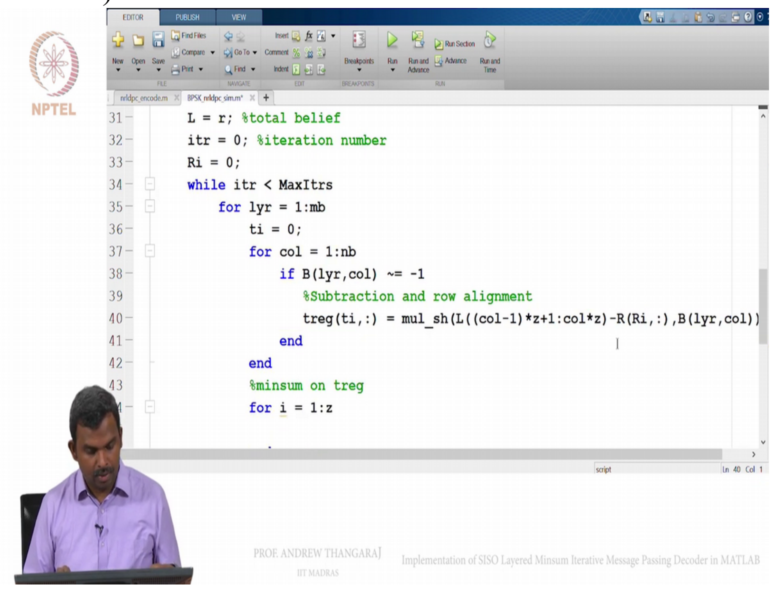
And, just for, Ok so maybe this is, this is slightly, maybe we will keep it as 0 here just for,

(Refer Slide Time: 31:30)




I will do this

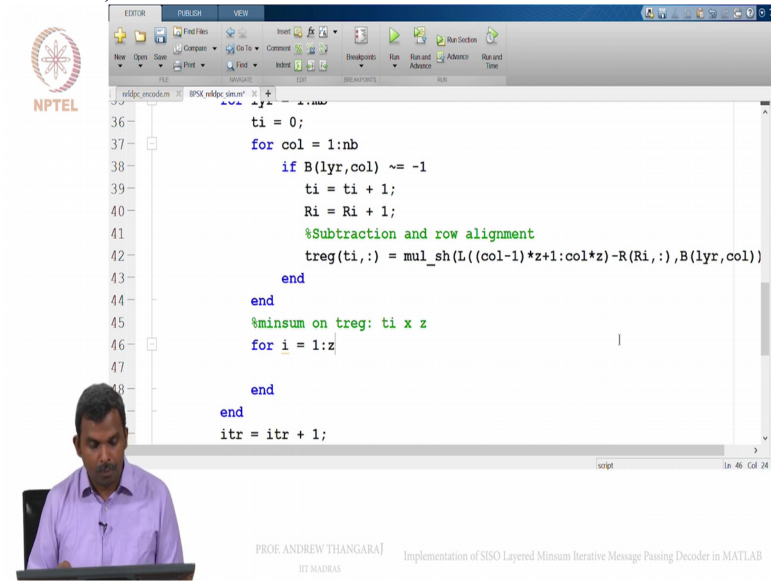
(Refer Slide Time: 31:37)




in a slightly reverse way, Ok. So this is better because if I start at 0 and add 1 before doing this, then I know t i is the total number of non-zero entries at that point. So I do not have to worry about that. So t i will be my total number of non-zero entries, Ok.

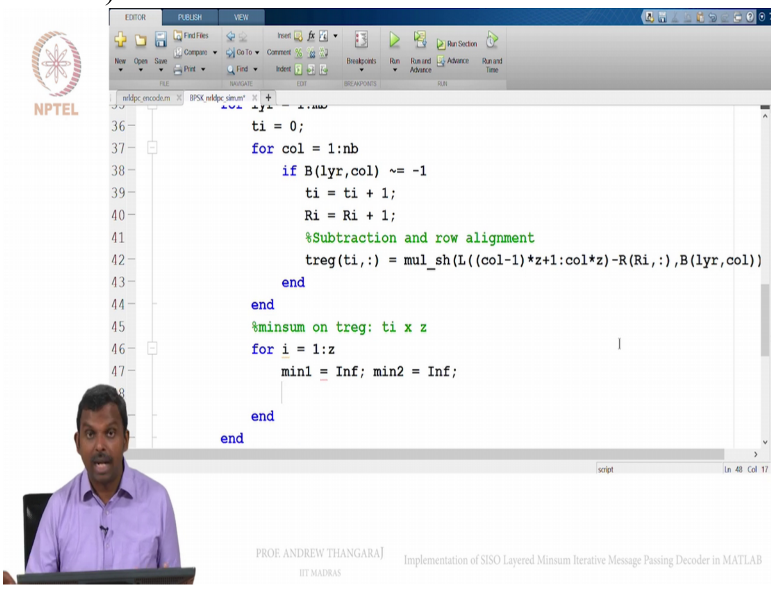
So t i, this is t i cross z, Ok.

(Refer Slide Time: 31:59)




So there are t i values and for every value in every, and that is all aligned nicely, so I just have to do the, do the row processing, Ok. So how do I, how I go about doing this? So so that is clear, so I need my min 1 and min 2, so I will set min 1 to be infinity, min 2 to be infinity. MATLAB allows me to do that,

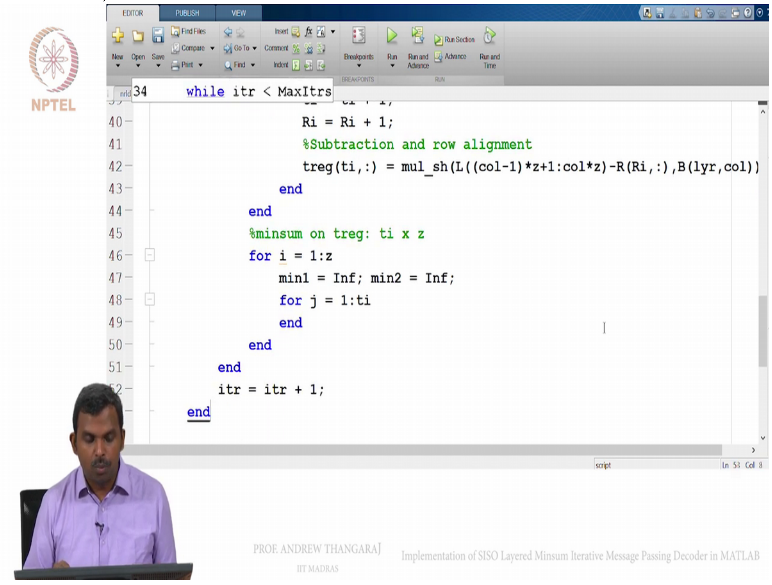
(Refer Slide Time: 32:27)




so large number Ok.

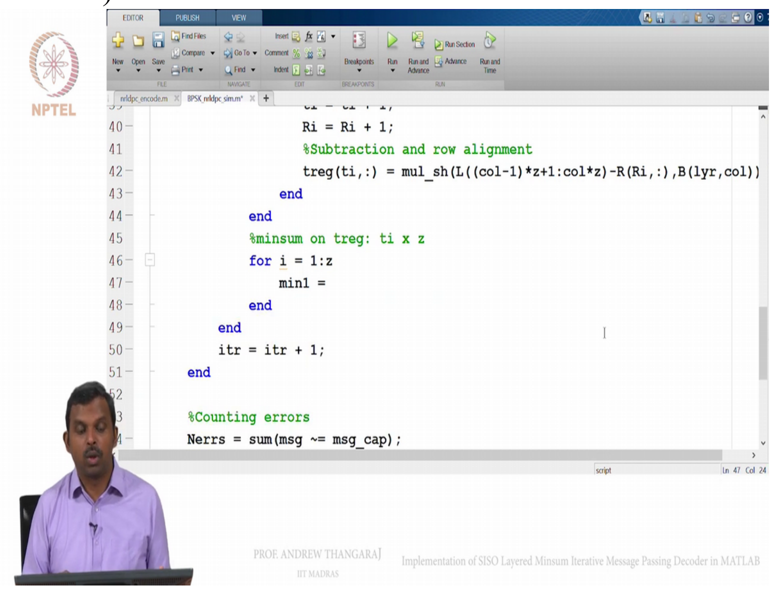
And then I will just process, Ok. So how do I process for j equals 1 colon t i? I will end it.

(Refer Slide Time: 32:38)




So loops actually are quite slow and bad in MATLAB so you should try to not use loops but I am just using loops for simplicity. So maybe we can avoid this loop here. So let us, write it that,

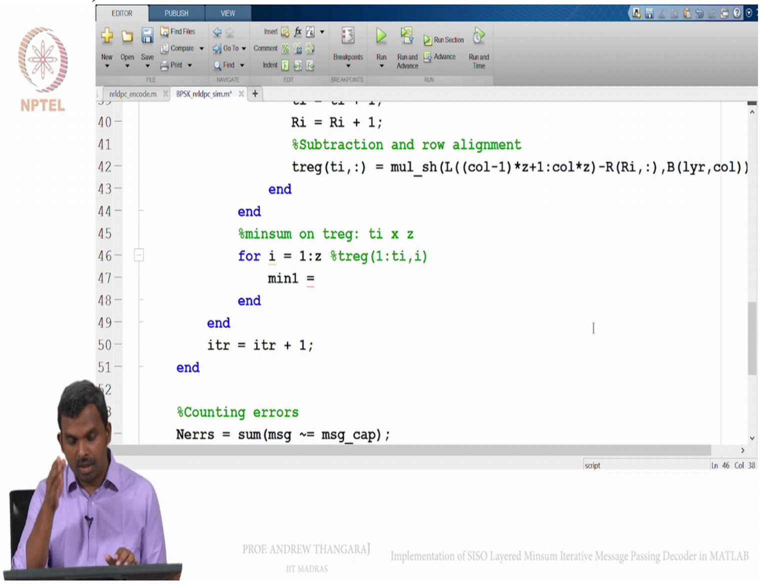
(Refer Slide Time: 32:53)




Ok.

So now I need to find, so remember I am processing the ith, so t reg of colon comma 1 colon t i comma i, Ok so the ith column

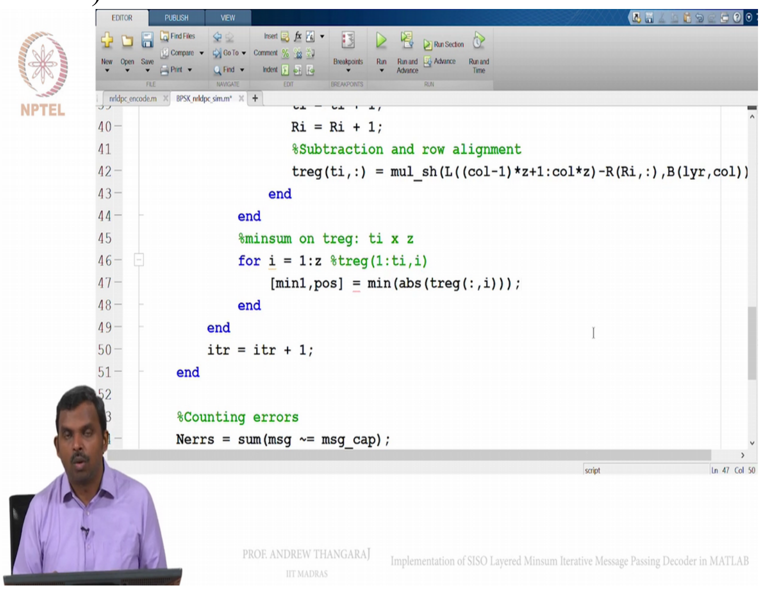
(Refer Slide Time: 33:08)




in t reg is what I am processing, Ok. So I have to find the minimum value among, absolute minimum value in the ith column of t i and then the second absolute minimum value then the product of the signs and then I flip based on that, right.

So that is what. I need to find the min. And not only the min, I also need the position, right. pos equals min of a b s of t i, t reg of colon comma i,

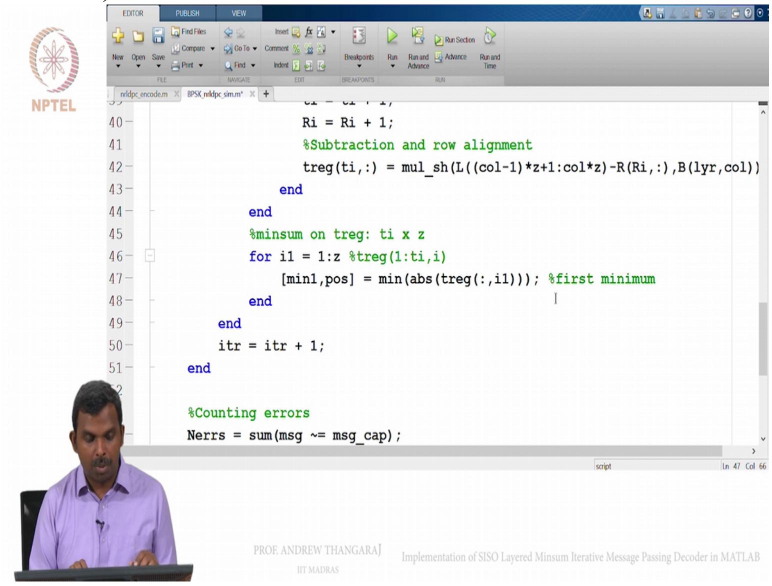
(Refer Slide Time: 33:43)




Ok.

So this gives me the first minimum. Is that Ok? So the first minimum, otherwise it is complaining over i, oh, Ok sorry it is i 1. So looks like I have used another index for i. There is an i for the ith block. So alright. So i 1 is something, Ok so this is the first minimum,

(Refer Slide Time: 34:13)

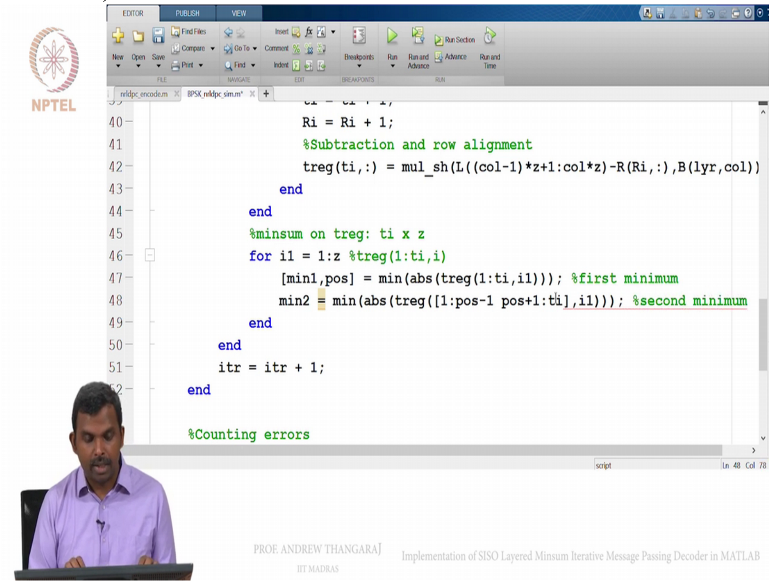



absolute minimum of course.

So I am ready to find the second minimum. For the second minimum, one should not use the value in pos. So min of abs of t reg of, I also not supposed to use the first value, so 1 colon pos minus 1 and then pos plus 1 colon end, not end, colon t i so it needs to be careful here, let us not use 1 colon t i, it is very important; comma i 1.

Ok so this should give me the second minimum. We will check it out if I made a mistake,

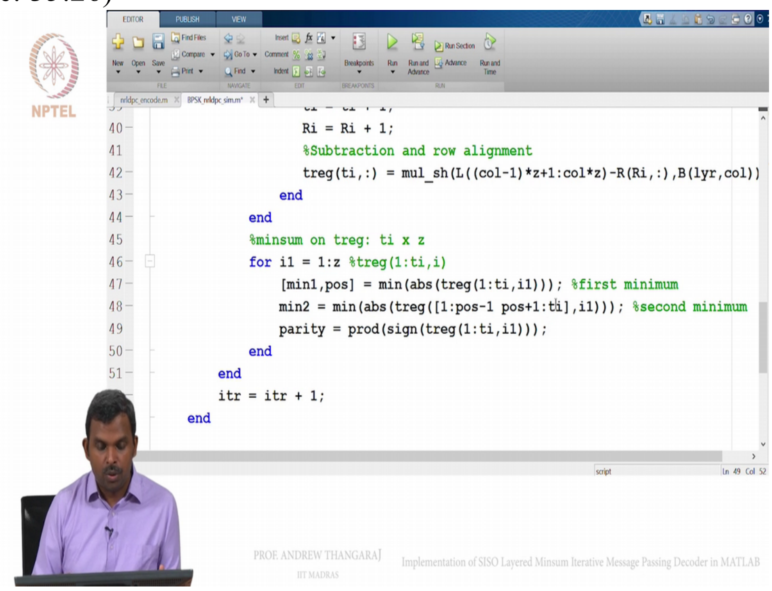
(Refer Slide Time: 34:52)




we will correct it. So this should give second minimum, Ok.

So the first minimum and second minimum are set, it is very nice, but then what about the sign? The sign, the sign is I need this parity. I take product of sign of t reg of 1 comma t i comma i 1, Ok. So that is parity for you and then

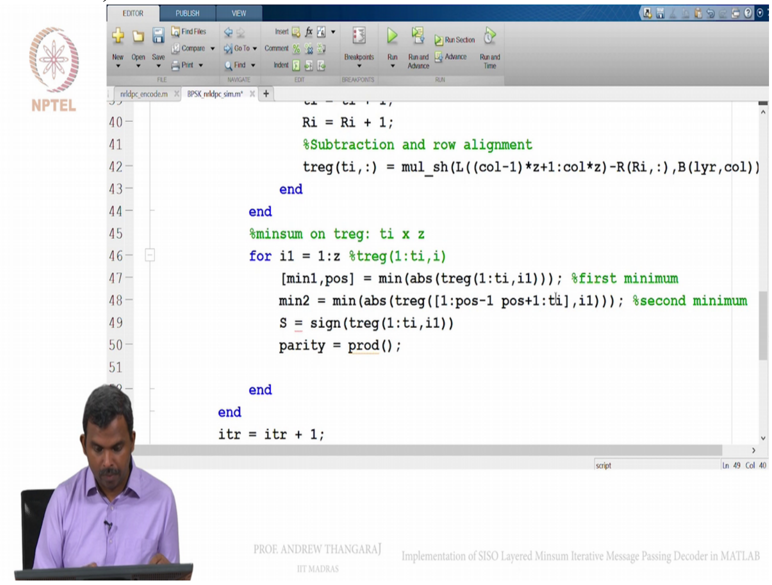
(Refer Slide Time: 35:26)




one can start, so it is good to keep the signs also.

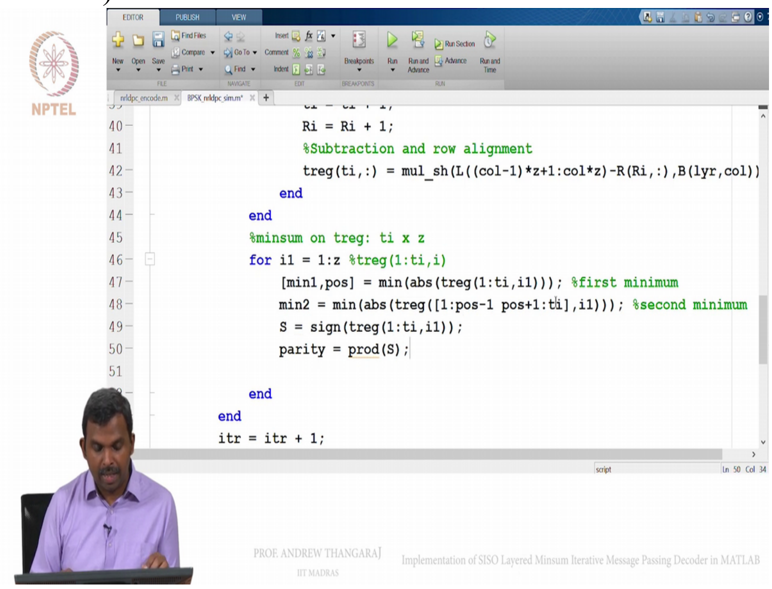
So may be what I will do is I will store the signs separately. I will call it S.

(Refer Slide Time: 35:55)



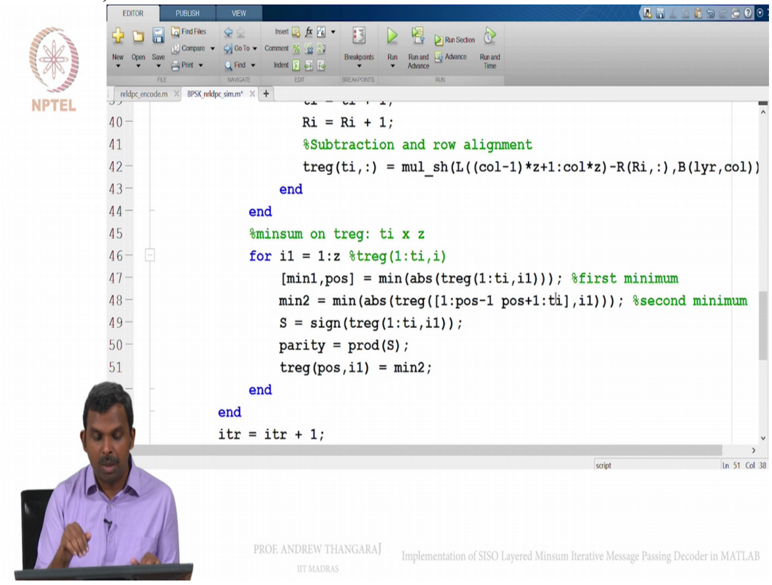

These are the signs. Product of S is the parity,

(Refer Slide Time: 35:58)




Ok and then t reg of pos comma i 1 equals min 2, Ok some setting

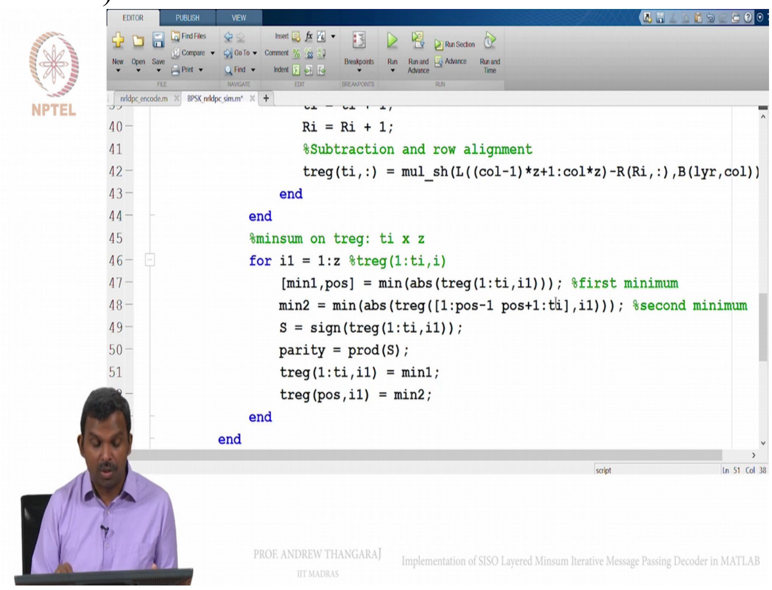
(Refer Slide Time: 36:10)




wherever the minimum was, I am setting the second minimum, t reg of, so actually it is better to do this in the, in the wrong, in the right way.

First do min t reg of 1 colon t i comma i 1 equals min 1. Let us do the absolute minimum

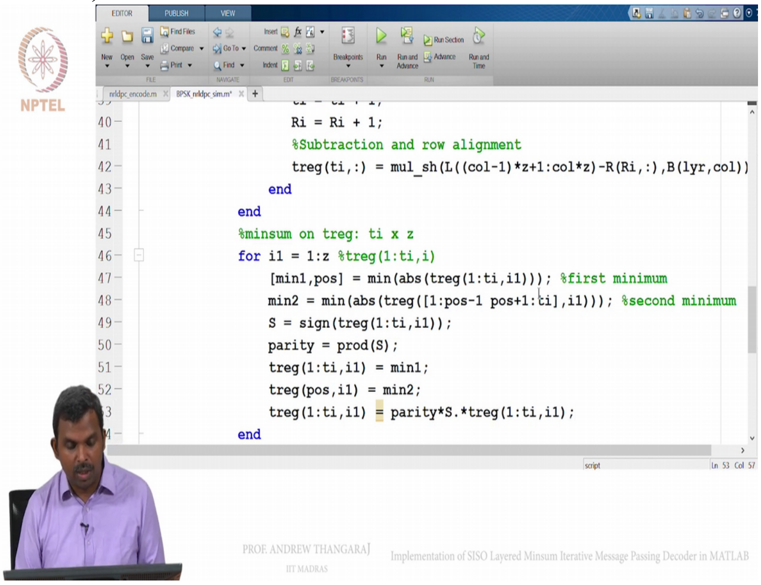
(Refer Slide Time: 36:28)




and then position alone you replace with min 2. So this is slightly easy way to do it. And what about the signs? t reg of 1 colon t i comma i 1 equals s prod parity times s dot into t reg of 1 colon t i comma i.

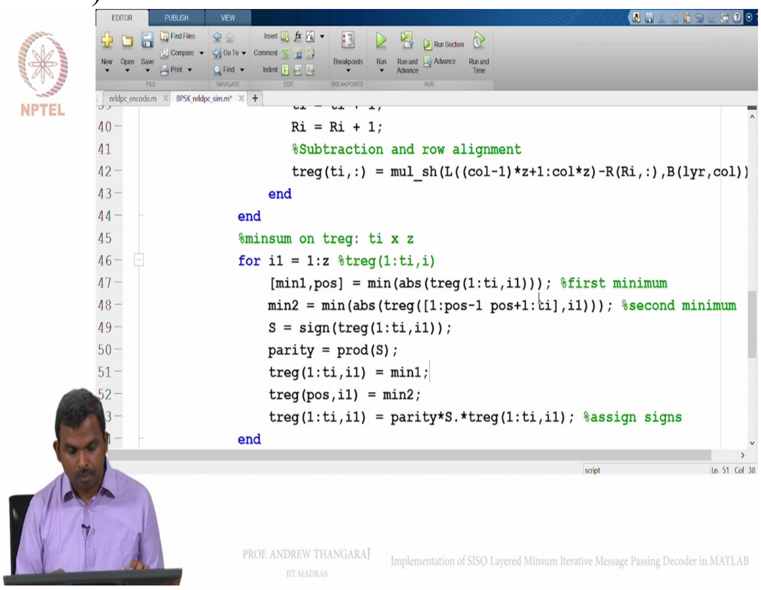
So first we assign the absolute

(Refer Slide Time: 36:53)

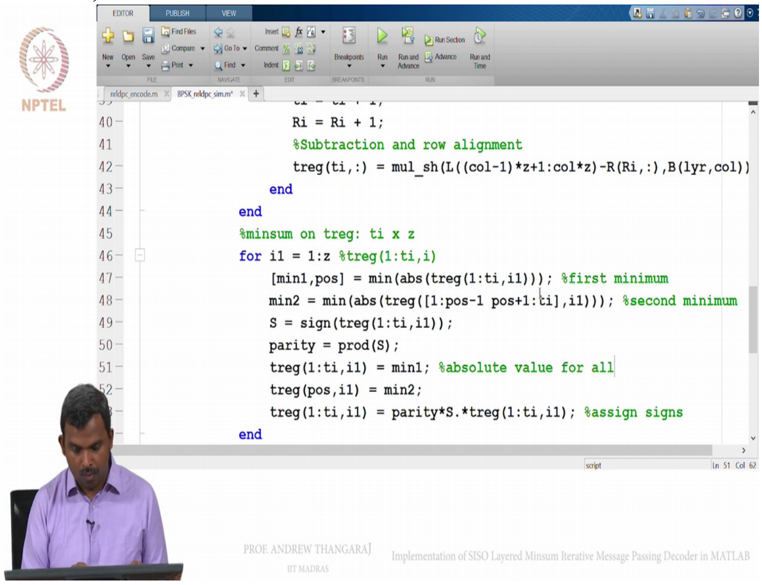



values, Ok so assign min 1 to everything and then the minimum, first minimum position gets replaced by min 2. So at this point only absolute values are there. And then we assign the signs, Ok.

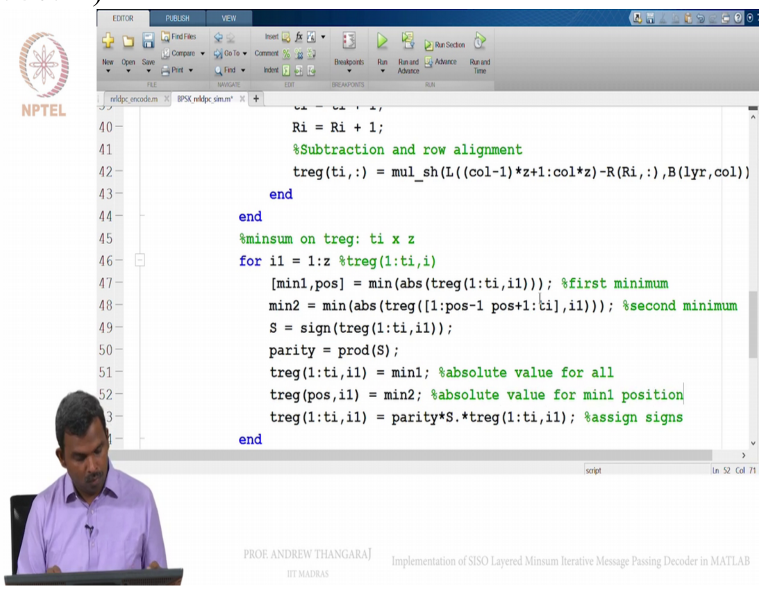
(Refer Slide Time: 37:07)



(Refer Slide Time: 37:15)




(Refer Slide Time: 37:22)

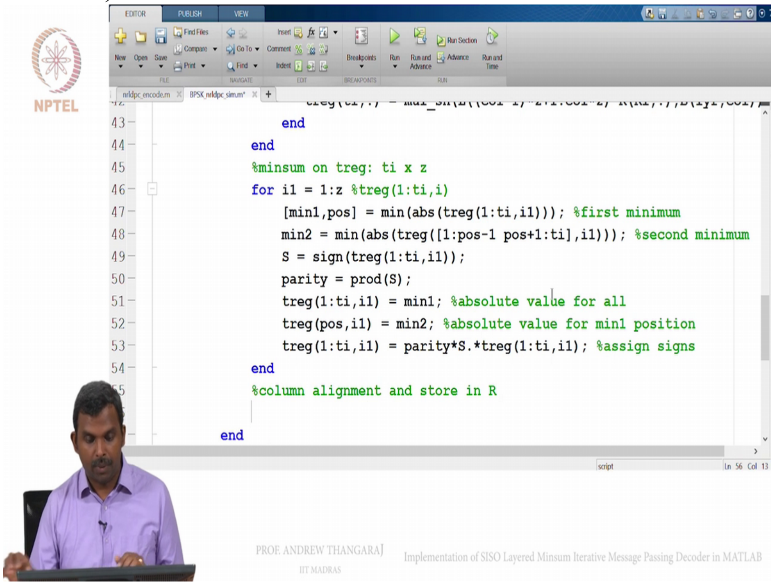



Ok, so hopefully this is clear enough. Remember once again we did subtraction and row alignment and stored the values in t reg. And now we just have to process the values in t reg for minsum operation. How do we do this?

We take the t i values and do minsum on each of those values, Ok. They are already aligned now so we do not have to bother aligning here. We just do, for each, each of the expanded position, expanded rows we do minsum, Ok.

After you have done with minsum, you have to write back, Ok. So column alignment and store in R,

(Refer Slide Time: 38:14)

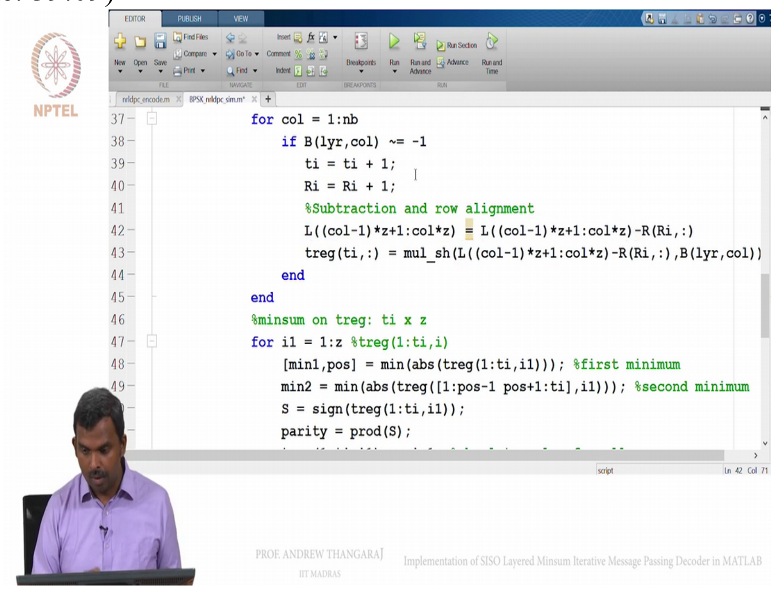


Ok. So for that there is an addition also, column alignment, addition and store in R. So one needs to be a little bit careful here. Remember, see when we did the subtraction, the subtracted values, Ok

So we did row processing on the subtracted values but then after the row processing we should not be adding into L. We should add to this. So, so one of the things to do here is, what are the things to do here which I did not do is the assignment of the subtraction.

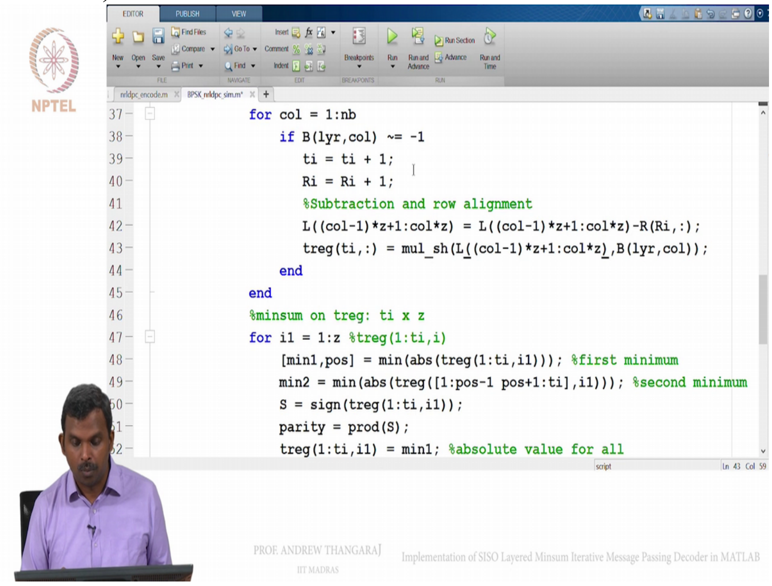
So I think it is good to do this. Let me. So this is important.

(Refer Slide Time: 39:09)



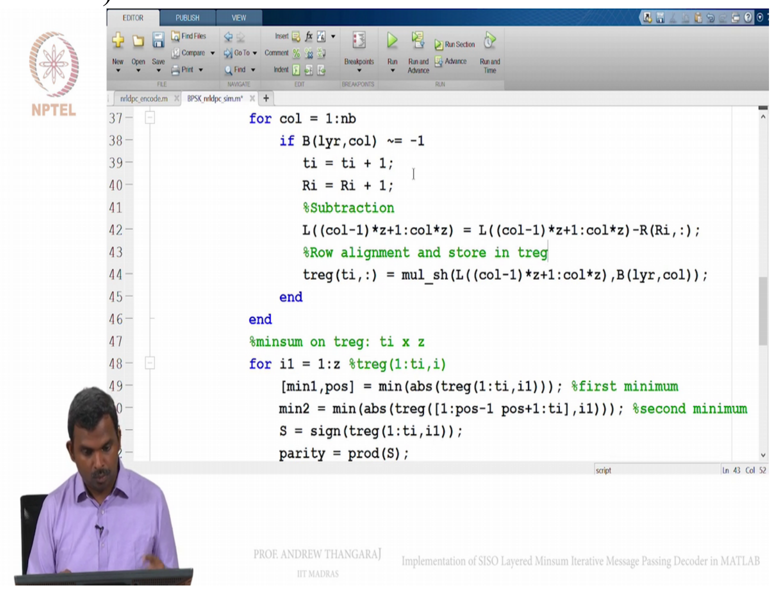
We need to do this. So once you subtract, you do not have to subtract again, Ok.

(Refer Slide Time: 39:15)




So this is nice, Ok. So we did subtraction and then row alignment goes here.

(Refer Slide Time: 39:32)



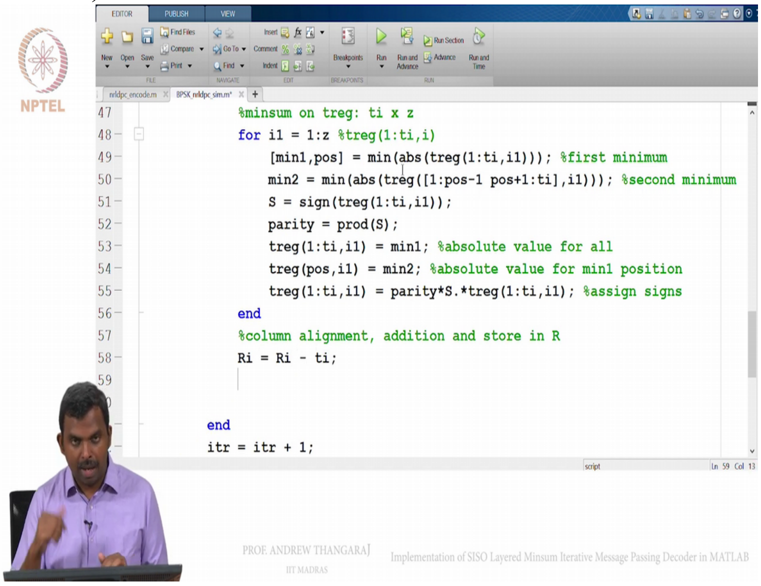

Ok so this is pretty clean. So our total belief has now been subtracted with R and we stored it in, we did the storage in, in the t reg. And then we did minsum. And once we did minsum, we can now add, Ok, column align, add and store back in R.

So I think this is the layer decoding algorithm. Hopefully this part is clear. So it is doing, it is not doing the subtraction and storing carefully. So I think this should be Ok, alright.

So now we do the column alignment and add and this needs to go on for every column, right. So this is, this will go like, Ok. So I think, this is, this is fine. Ok. Ok. So, so we will do the index again.

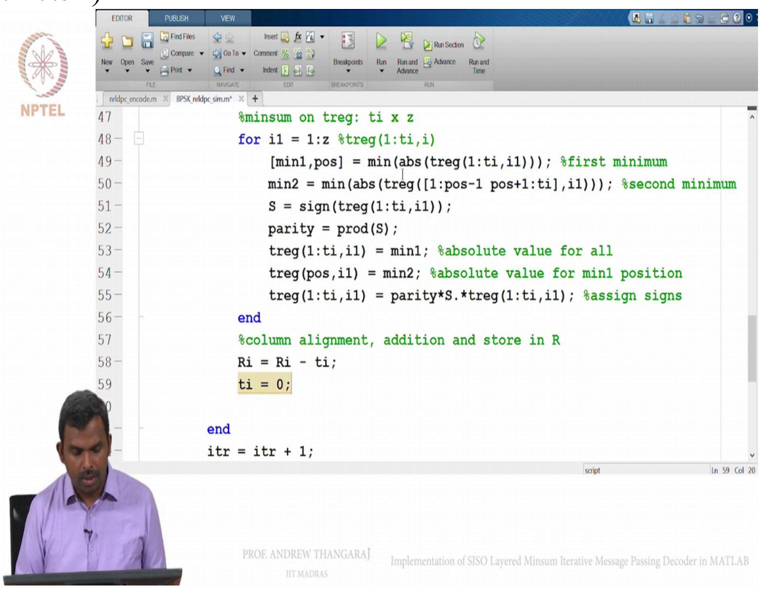
So R i is going to be R i minus t i.

(Refer Slide Time: 40:45)



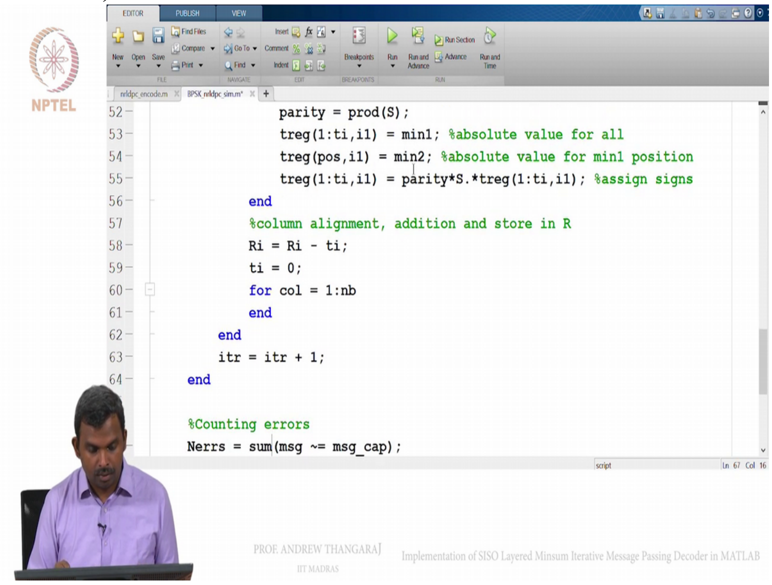

Ok snap back at the first index. t I, we need t i

(Refer Slide Time: 40:51)



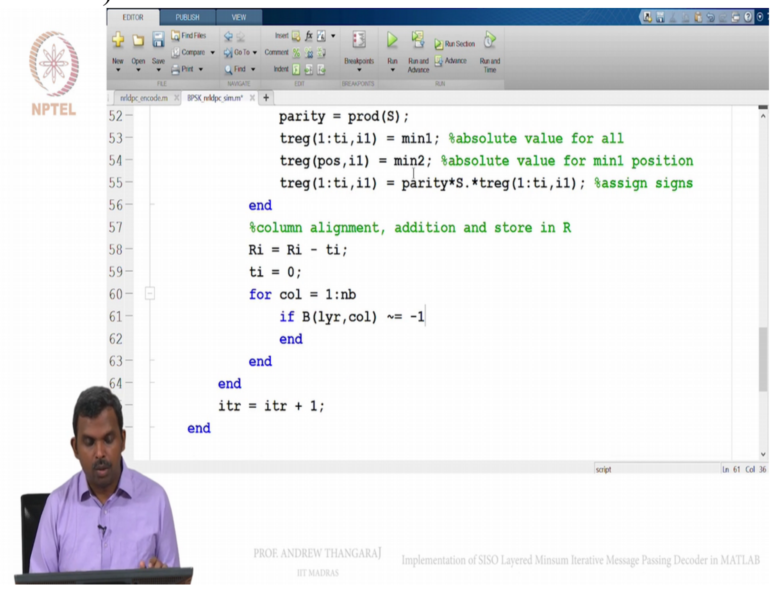

to go to 0. Let me see this, for column equals 1 colon m b and

(Refer Slide Time: 40:59)




if B of layer comma col not equal to minus 1, then you can

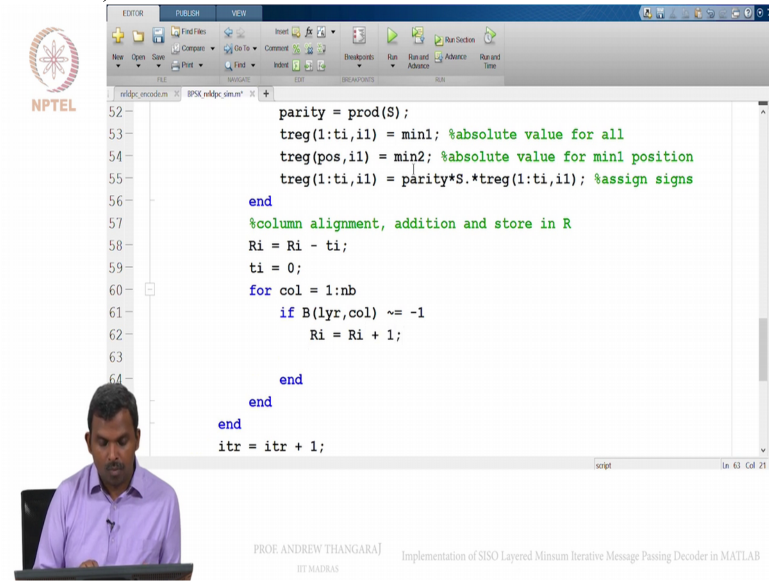
(Refer Slide Time: 41:09)




do something here.

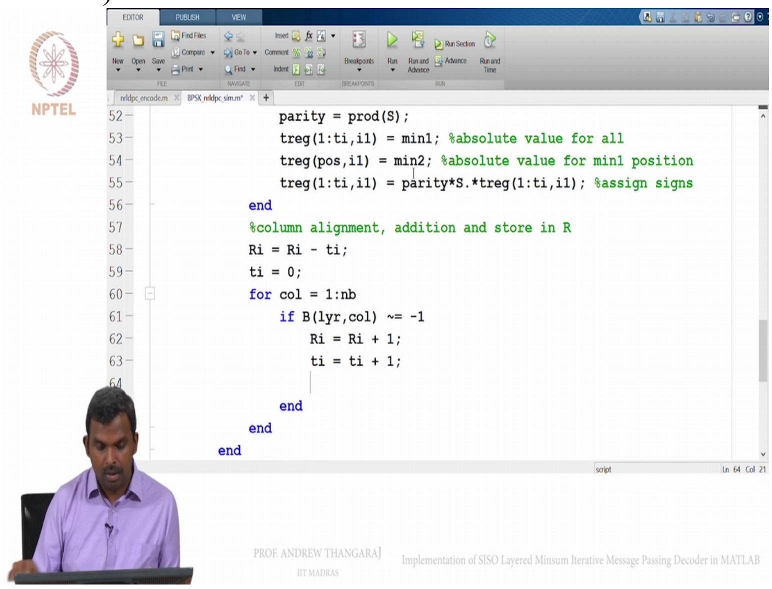
So what do we do? I need to align according to the column and add it to L, right. So that is the, that is the important part there. So R of, R i equals R i plus 1.

(Refer Slide Time: 41:26)




t i equals t i plus 1.

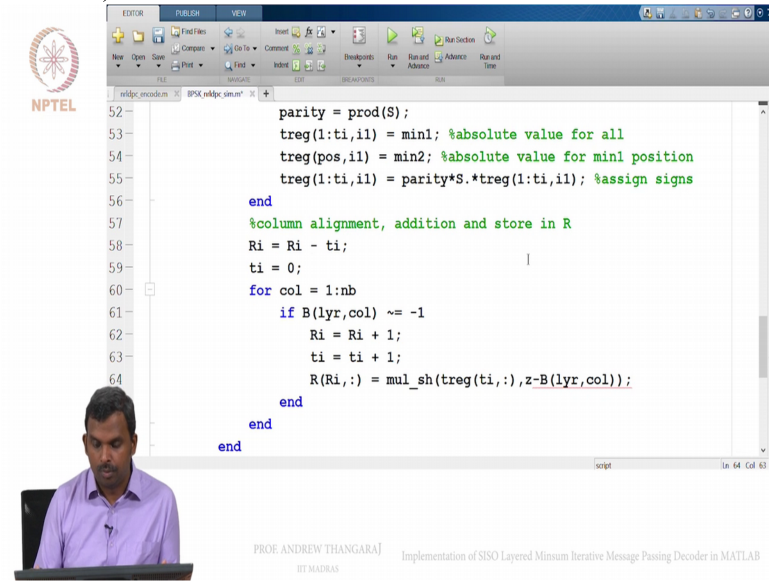
(Refer Slide Time: 41:31)




Then I store back. So R of R i comma colon equals, so the, so, so the alignment needs to change here again. So I pushed it, I did the subtraction correctly and then I shifted it so that I got the row alignment done.

Now I have to do the inverse of the shift. So try to think about that. So I have to reverse the shift and then align it and then store it in R, Ok. So R itself needs the alignment. So I need to mul shift of R of, of sorry t reg of t i comma colon, right with z minus B of l y r comma c o l. Ok

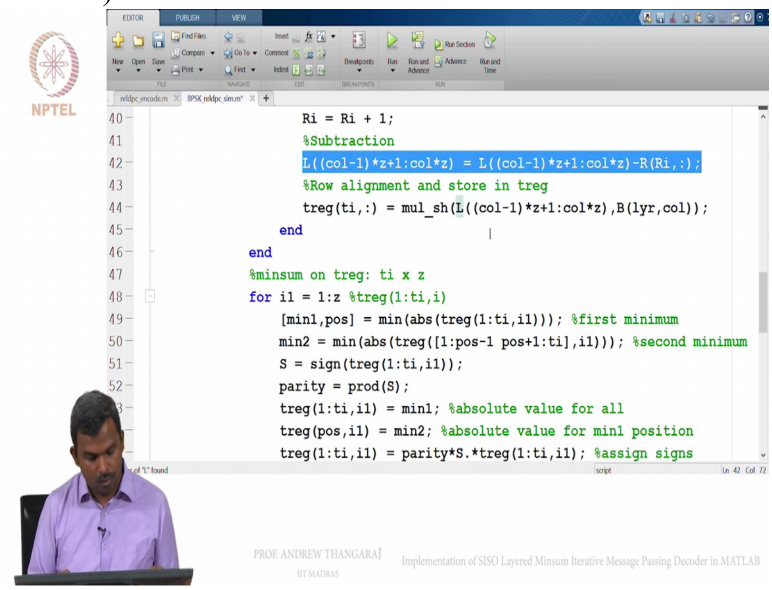
(Refer Slide Time: 42:30)




so this is the inverse of the rotation and that gets stored in R.

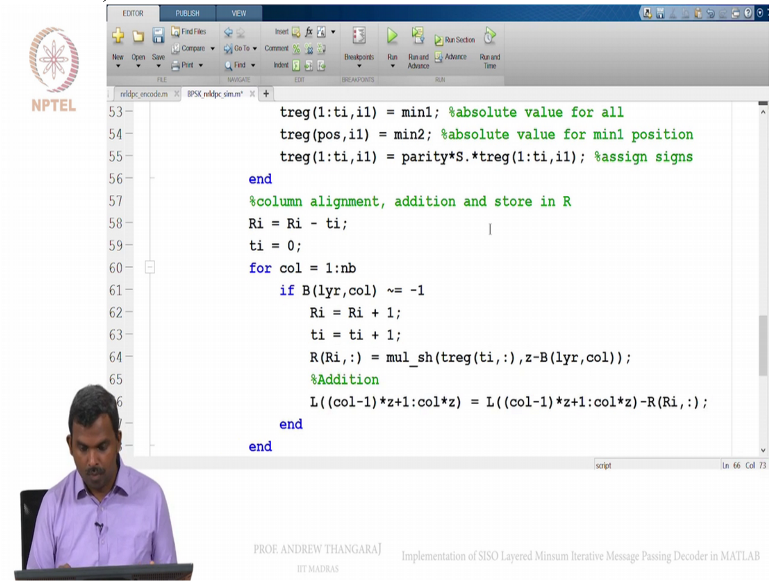
Once you have stored this in R the L process is quite Ok. So we can go and do a cut and paste here. We do not have to the whole typing.

(Refer Slide Time: 42:49)




Of course, this is the thing,

(Refer Slide Time: 42:59)

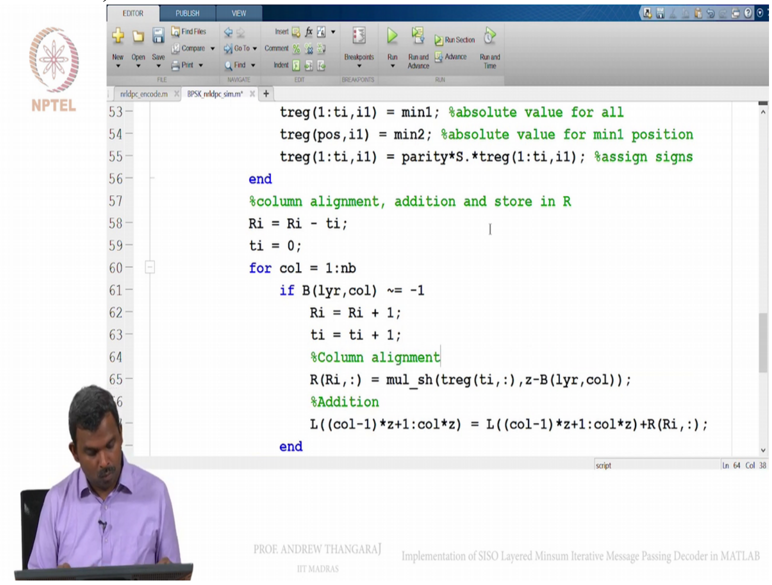



Ok. So think I have come to the end of the iteration over every layer. I think I am doing everything here. We will try it out and see if I made any mistakes here.

So the total belief is stored there. Your iteration number and you have R i which counts the storage and then I go through and do the subtraction and then the alignment I store in the temporary thing. I then run through minsum on the temporary thing and then I column align again

But remember I have to do this adjustment on R i. I have to go back to the original index in my storage. And then store it back. I think I am doing Ok here, Ok. So this looks fine to me. I have done the column alignment here,

(Refer Slide Time: 43:56)

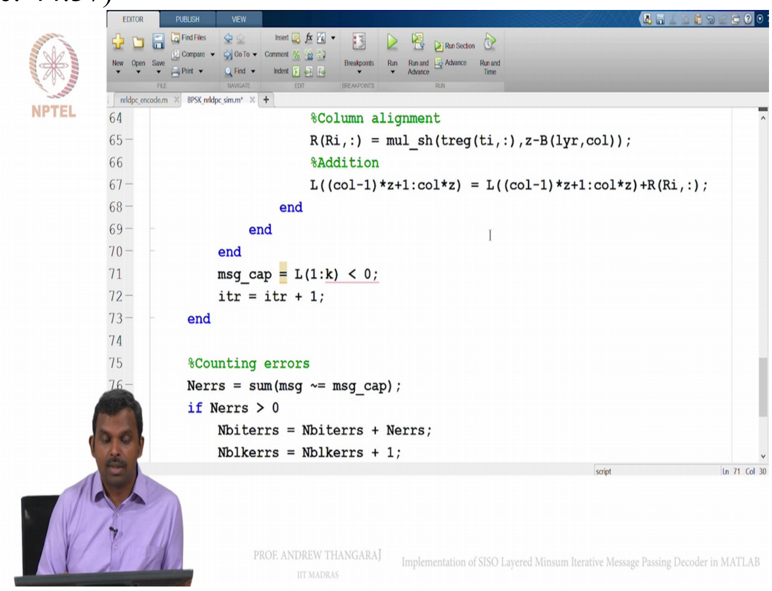



Ok and then I have added it to L and we are good to continue.

So once I am done with this, I have done my, done my first layer. The layer is over. I have done all three operations on the layer. Once all the layers are over the iteration itself is complete, Ok. So now when the iteration is complete I have to do message cap, Ok. So I have to make decisions. So how we make decisions?

We know how to do this. message cap is l of 1 colon k, right the first k bits are the message bits and if they are less than

(Refer Slide Time: 44:37)




0, it is 1, if they are greater than 0, it is minus 1, Ok and that is the end of the iteration.

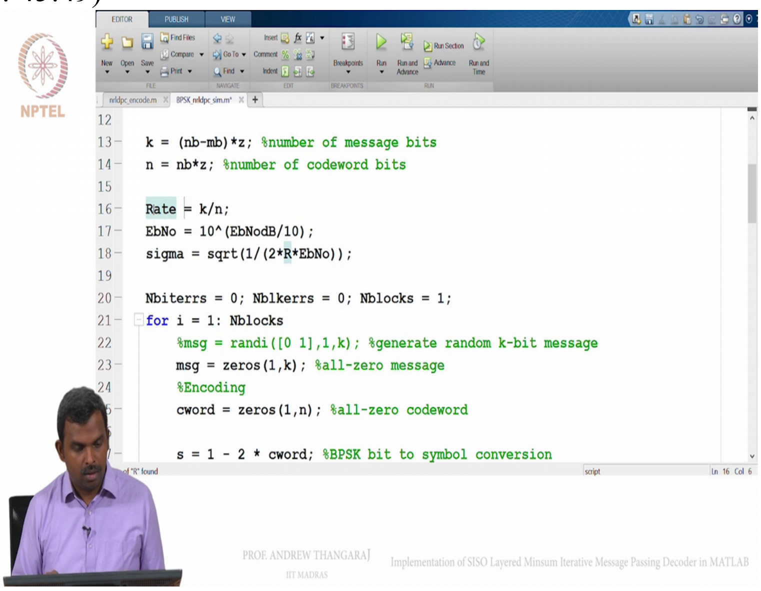
Once the iterations are done we come out and check if message is not equal to message cap, Ok. So that was the coding. Hopefully you saw the various things that I was doing. I am definitely not claiming that this is the most efficient way to code up your L D P C coder.

I am sure there are much better ways to code it. And if you are not interested in seeing how I was doing the coding you do not have to see it also. You can see the final code. It is available.

So let me just go through this code once just the way I have written it and we will, we will try it out. I mean I have to debug this. It should be another lecture but let me, let me show all the overall idea and how I have done that.

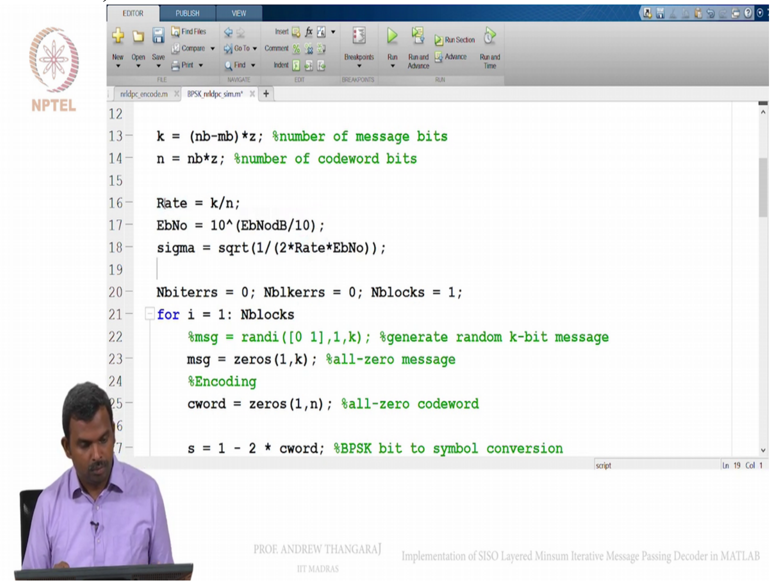
So you load the base matrix, take its size, and look at the expansion factor, number of non-minus 1s is important, that is the storage you need and then temporary storage. So there is some confusion about this R, Ok that is the rate, Ok. Can I redefine this as rate? Ok I will change this R as

(Refer Slide Time: 45:49)




Rate here.

(Refer Slide Time: 45:54)




So this Rate have any other role to play? Probably not, Ok so Rate is good.

So these are things to watch out for. If you reuse the variable MATLAB will warn you. That is a nice thing to have. So you can see that you are not making mistake here.

R is the storage for the row processing. t reg is the temporary storage k is the number of message bits, n b minus m b into z, n is the total number of codeword bits, Rate and you use Rate to find E b over N naught and sigma.

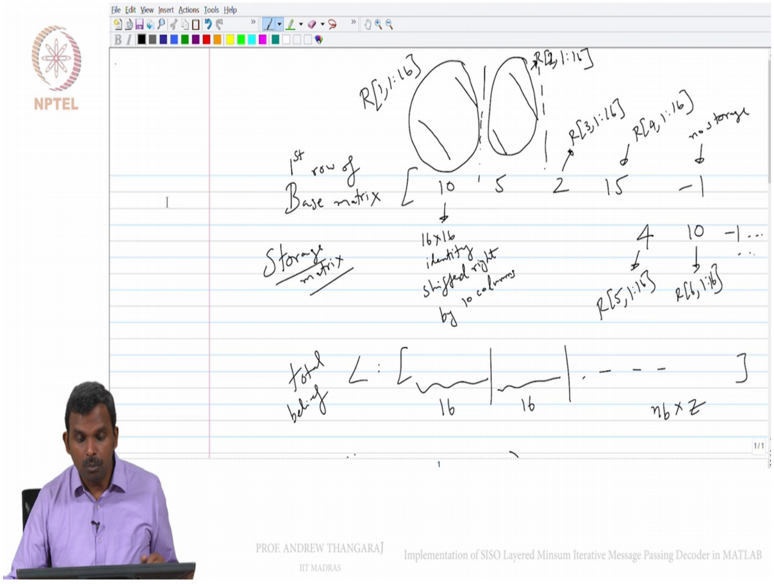
And then we are encoding the all zero codeword, so that is something, for simplicity we are doing that. For the decoder it is not so crucial. We do B P S K modulation as before and then add noise at standard deviation sigma.

Ok now I will start the soft decision iterative message passing layer decoding. You have your total belief L which is very important. Based on that you make decisions. Iteration number R i and I have outer loop for iterations, for every iteration you have to do this. We have to go through all the layers and for every layer you have to process all the columns, Ok.

So there are definitely much more efficient ways of doing these operations even in MATLAB but I am just doing it laboriously to show you how it works. You go through every column. Whenever you have a minus 1 you pull that value from the storage into the register but you do this subtraction first, L is subtracted and then you do a row alignment and store in the temporary register.

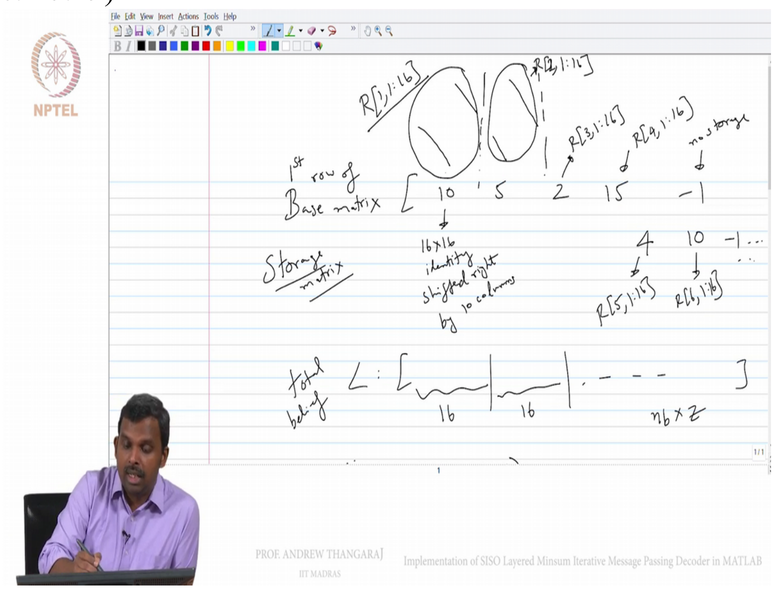
So the row alignment once again, if you are confused about why I need to row alignment,

(Refer Slide Time: 47:26)



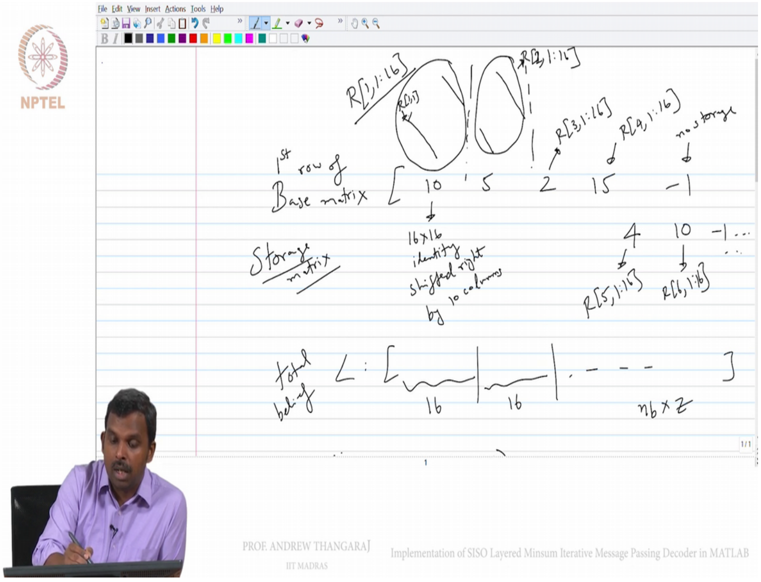

so if you see here, R is store

(Refer Slide Time: 47:29)



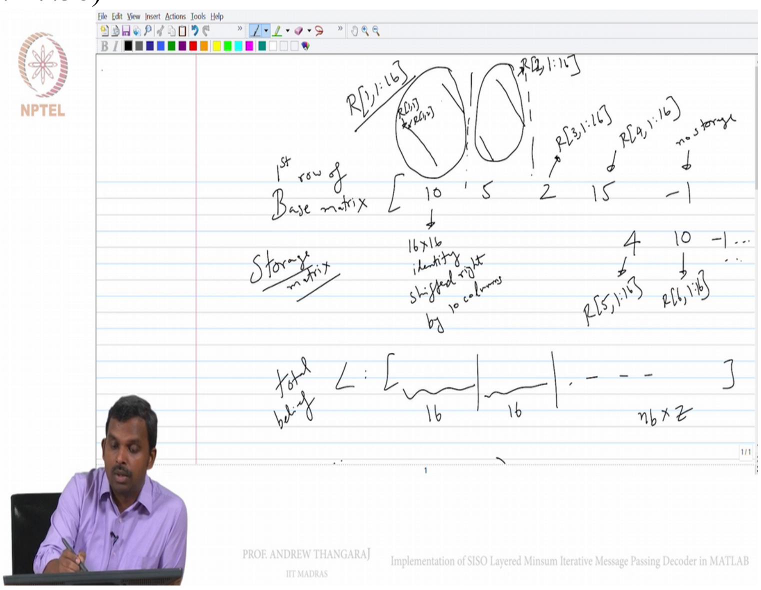

1 to 16 in this sequence. So this will be R of 1 comma 1, this will be

(Refer Slide Time: 47:36)




R of 1 comma 2

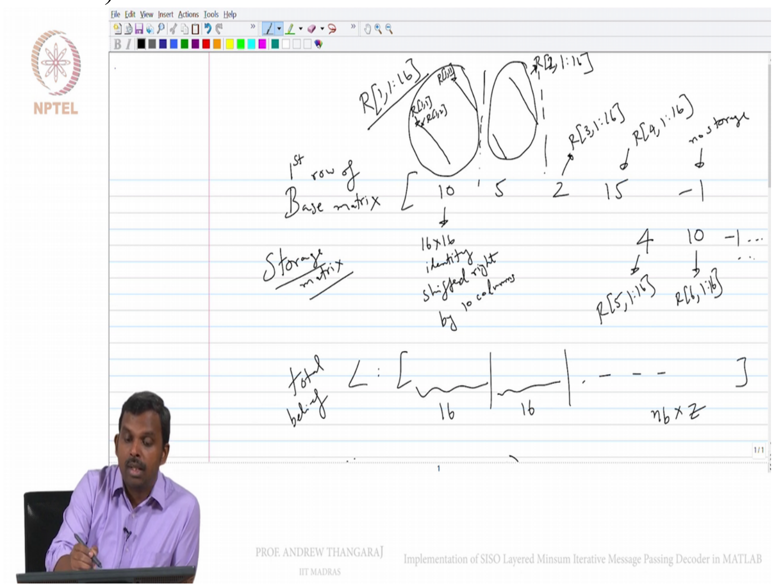
(Refer Slide Time: 47:38)




and so on.

Ok so this guy will be actually be, so this is 10, so this will actually be in MATLAB's thing, it will be R of 1 comma 11.

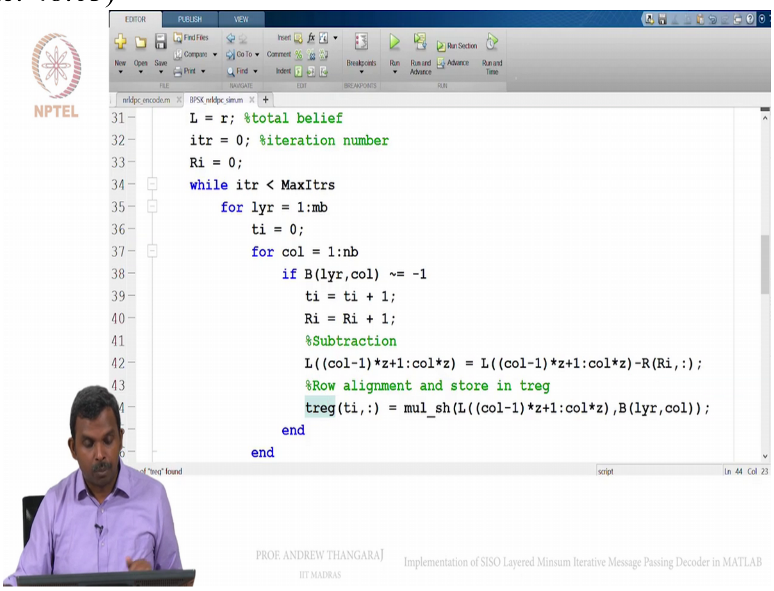
(Refer Slide Time: 47:46)




Ok. So R of 1 comma 11 actually corresponds to the first row after expansion. So R 1 comma 1 is not the first row after expansion. After expansion R of 1 comma 11 is the first row. So I have to align it so the first row matches throughout. So I can just go through .

So that is the alignment operation here.

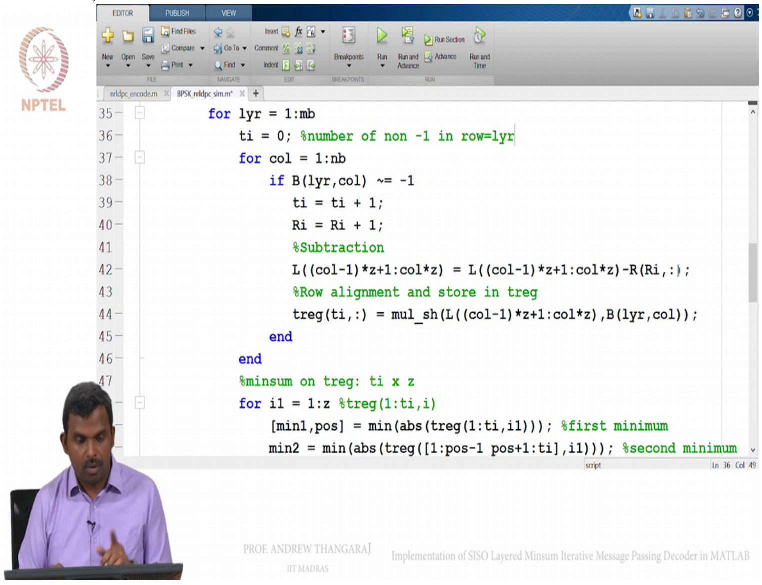
(Refer Slide Time: 48:03)




So we can see we have done the shifting by whatever value of B to get my, get my correct value done, Ok. So, I am a little, yeah I think this is Ok. So subtracted, and that is the value I am storing into my storage register, Ok.

So that is good. And once I, once I did that I have to do minsum. Ok and minsum I just do on t reg and this t i I have kept count. This t i is the number of non-minus 1s in each row.

(Refer Slide Time: 48:47)




Ok and so when row is l y r; t i is track of the number of non-minus 1s. So only the t i by z matter in t reg, there can be 19 rows, non minus 1 entries in each row at most so I am keeping it as t i by z, Ok.

And then I have this i 1. So I am processing the i 1th expanded row, there are z expanded rows in every block row, and the i 1th row I am processing one row after the other I am going.

I find the first minimum, absolute value right in the i 1th column, then the second minimum in the absolute value; I also need the position of the first minimum. And leave out the first minimum position and find the second minimum, right? So that is what I do here.

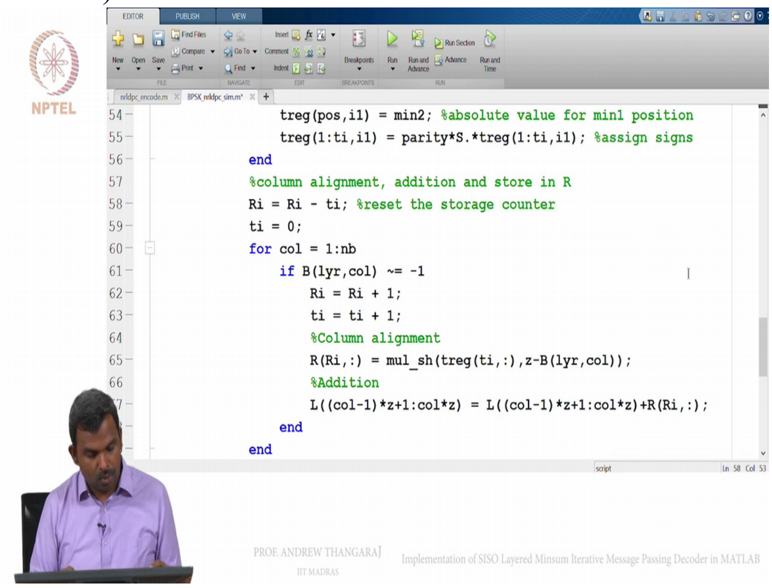
And then all the signs and then the product of the signs, the parity Ok. And then I assign the absolute value first after the minsum processing. Everything is min 1 except for the pos which will be the position of min 1 which gets replaced by min 2. This is just an efficient way of doing it very quickly.

And then, the sign is a sign, right. You have to multiply with the sign. I used the dot into here because both of these are vectors, you have to do dot into, Ok. And then we are ready to write back, Ok. So for that you need column alignment again.

So you remember t reg is row aligned and I have to make it column aligned, Ok. And when I have to make it column aligned I have to do the reverse of the row alignment rotation that I did, Ok.

So I have the other, so I am going back. This is reset the storage counter, Ok.

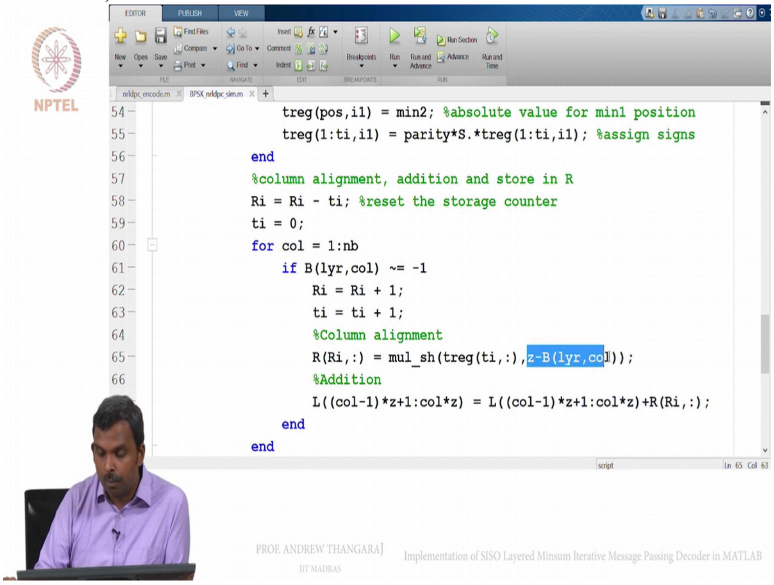
(Refer Slide Time: 50:28)




So we have come through, to the end of the row. You have to go back, reset it and then do that. Then reset the counter for the non-minus 1 entries and then go through the increment R i by 1, and this is my column alignment shift.

So you see instead of B of layer comma col, I am doing z minus B of layer comma col, right. So that is important to note,

(Refer Slide Time: 50:49)

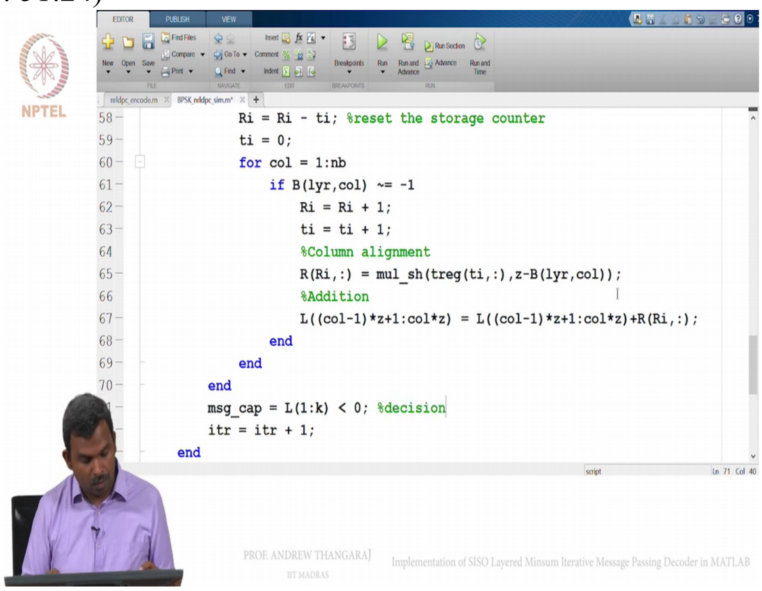



Ok. So this is t reg of t i comma colon and then I add, Ok. So I take the old values such that, L and then I add it to R of R i comma i.

So I think this is correct. Think I have done everything correctly but when we start debugging we will know if I made any mistake, minor error or what. May be you caught some minor error I made. Once we finish debugging we will have our code ready, Ok.

So remember total belief is the actual final belief you have based on which you have to make a decision. And that is the decision here. Ok so it is easy to do.

(Refer Slide Time: 51:24)




Ok.

So lot of people will design on the entire codeword and use the parity check condition to see if the codeword is valid or not and exit earlier than the maximum number of iterations. I have not done that here. I am just showing you the whole iterations.

May be if you want you can add that later on. So that is why there is a while loop here, Ok. So once you come out, after that everything is the same, right what you did for the Hamming code, you have to do here. Count the total number of errors.

If there is any error at all, you increment the block errors by 1. And the bit errors get added on to nbit errors. Find the bitrate, frame error rate and divide. Nothing changes there, Ok

So hopefully I mean this is a little bit of a code. I mean, it is not very easy. It is not that complicated either, right. So the storage is what you have to manage carefully and, believe me this is not the only way to do it.

There are very, very efficient ways to write this code. In fact you can write this code without any for loops, Ok. So you can write it very efficiently in MATLAB using MATLAB's sparse matrix structure. It is possible to do these things.

But I am not doing all that right now. So I am just, this is more pedantic in the sense. I am just trying to explain how the operations go and focus on the operations as opposed to efficiency in code, Ok.

But still we have not debugged this yet. We will have to debug it and see how it works. We will do that in the next lecture. Thank you very much.