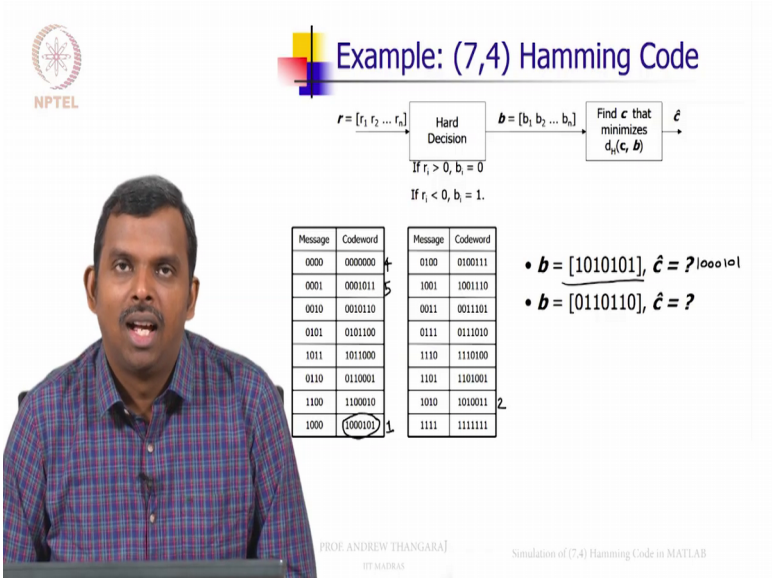
**LDPC and Polar Codes in 5G Standard**
**Professor Andrew Thangaraj**
**Department of Electrical Engineering**
**Indian Institute of Technology Madras**
**Simulation of (7,4) Hamming Code in MATLAB**

(Refer Slide Time 00:16)

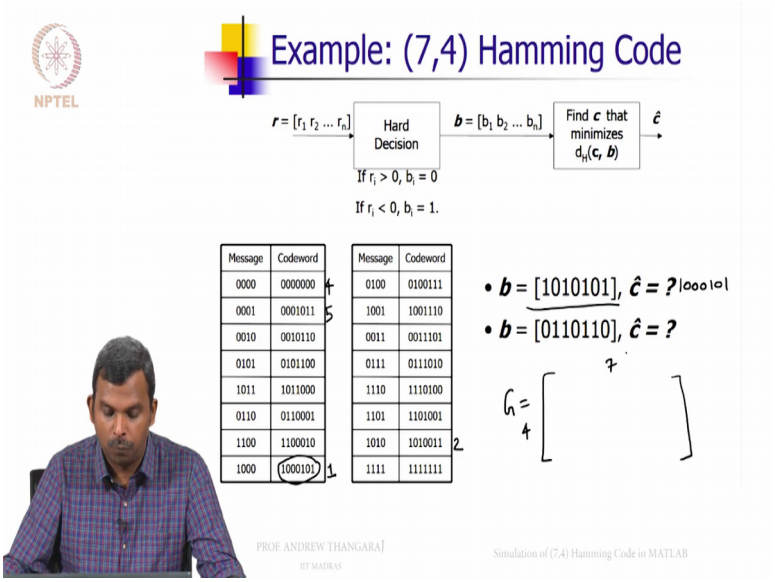



Hello, so in this lecture we will be looking at building decoders for the 7 4 Hamming code. You remember in the previous Matlab coding lectures we saw how to build decoders for the repetition code. We built both the soft decision decoder and the hard decision decoder. We ran them and saw what the output was and compared it with some plots and we got good answers.

We will do the same for the Hamming code. I will build it very quickly and show you. But a couple of things I want to point out once again, remind you. The code I write primarily using Matlab for the demonstrations, you can use Octave as well, and the same code that we provide for you should work on Octave also, Ok.

So let us get started. So this is the Hamming code, the 7 4 Hamming code. So, I mentioned that this is a linear code as well and one can write down the generator matrix for this linear code. So this would come out like this. So you have G equals, remember it is a 7 4 code so it will be 4 by 7 generator matrix,

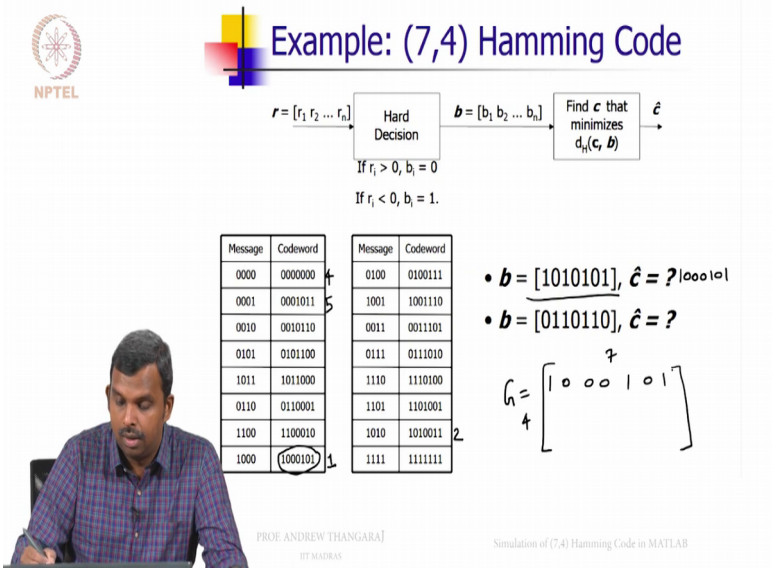
(Refer Slide Time 01:29)




Ok. So it turns out since, since this encoding is systematic you can quickly write down the generator matrix. So the generator matrix will look like this.

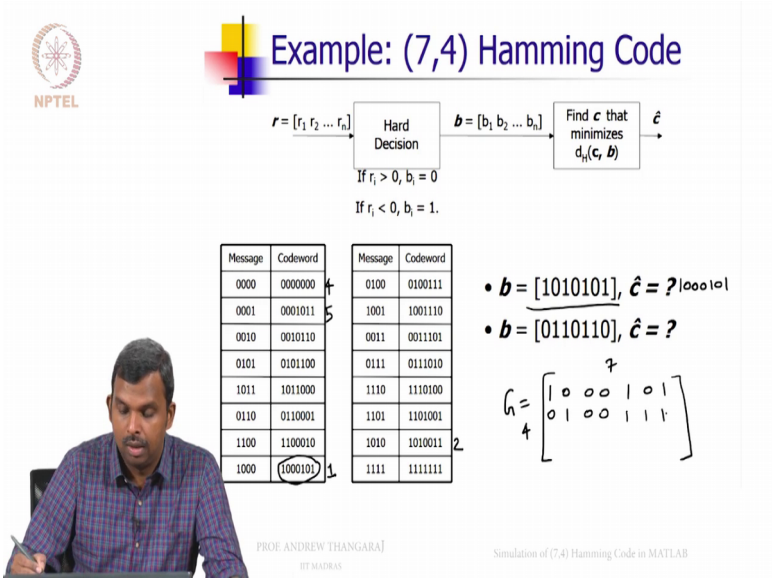
1 0 1

(Refer Slide Time 01:42)



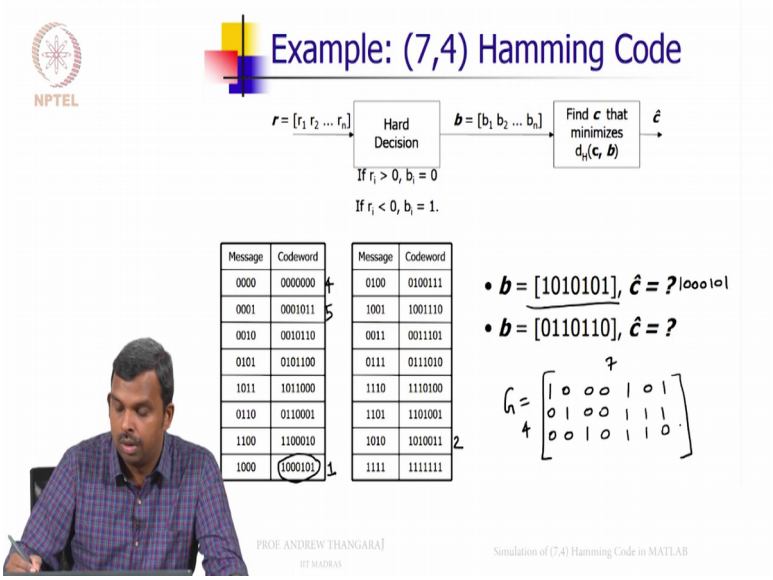

and 1 1 1

(Refer Slide Time 01:53)



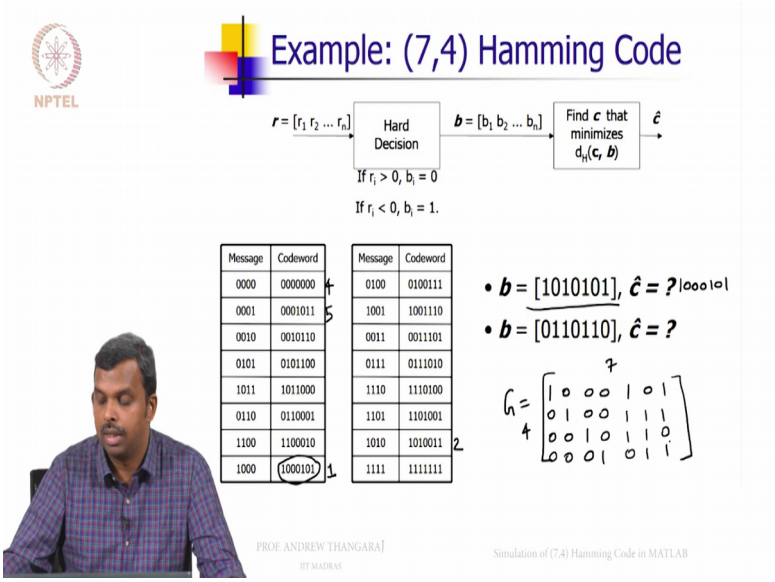

and then you have 0 0 1 0 and 1 1 0

(Refer Slide Time 02:00)




and 0 0 0 1, 0 1 1. Ok

(Refer Slide Time 02:07)



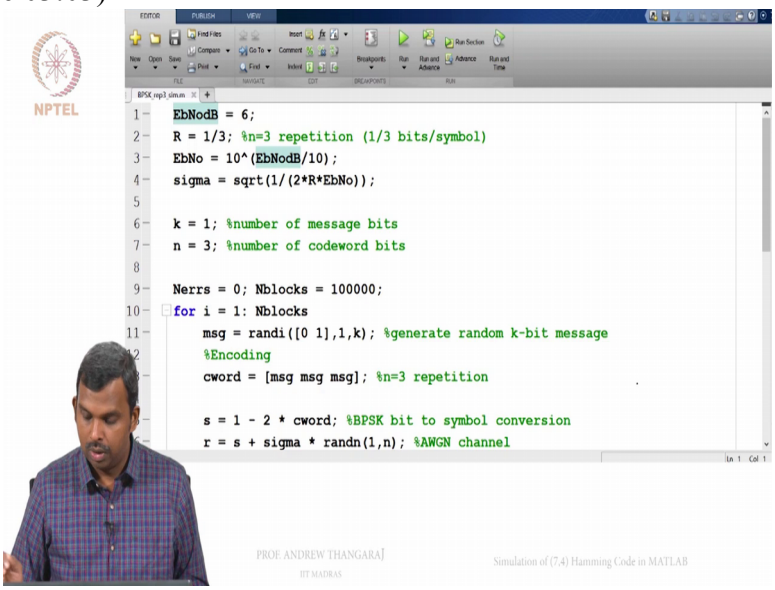

so this is the generator matrix for this 7 4 Hamming code. So you have, when you multiply m with G with the message bits appearing in the first 4 locations and then the first parity will be m 0 plus m 1 plus m 2. The second parity will be m 1 plus m 2 plus 3. And the third parity is m 1 plus m 2 plus m 4, Ok.

So this is the generator matrix. One can write down the parity check matrix as well for this, for this code so that will be just p transpose and I 3 (()). And one can use it, Ok.

So, so I am giving you this just for, just for completing the previous lecture but this you may want to use it in the, in your coding for instance when you want to do encoding this is a good thing to use, Ok.

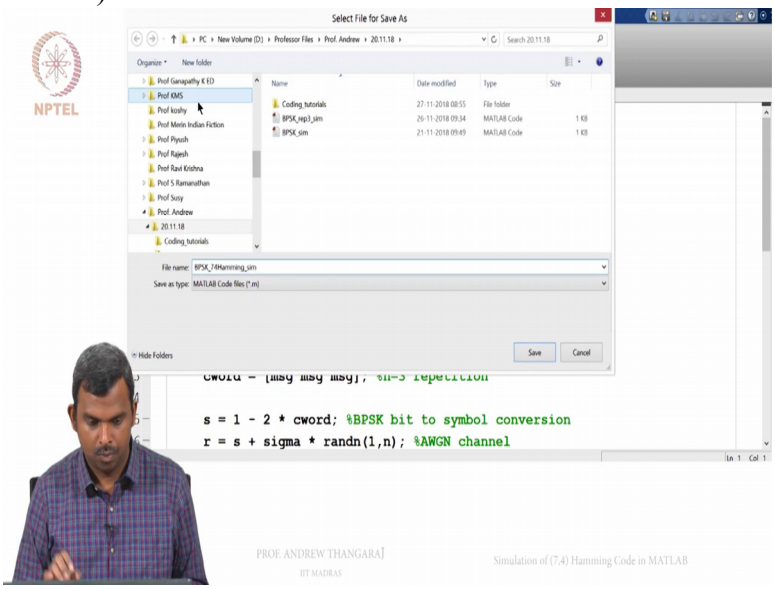
So let us move over to Matlab.

(Refer Slide Time 03:03)




So I am going to save this as 7 4 Hamming

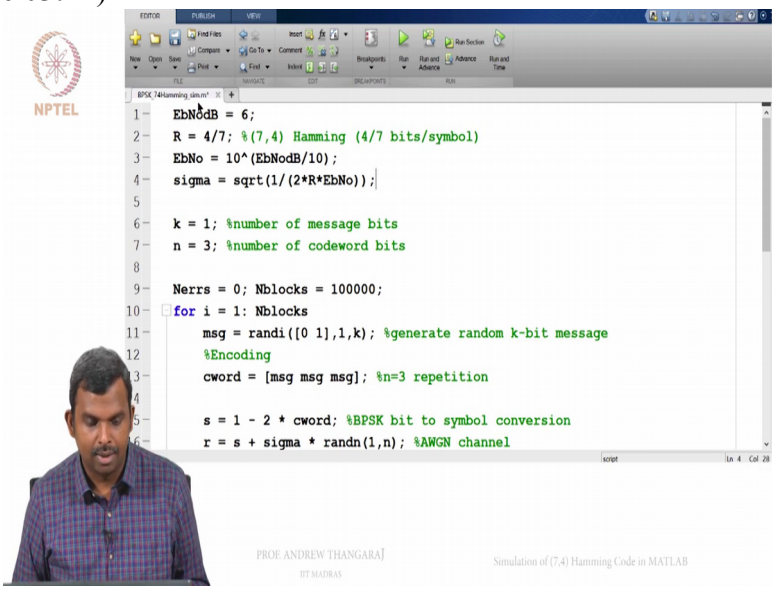
(Refer Slide Time 03:16)




Ok so we will use the same 6 d B, remember the rate is 4 by 7 Ok so let us make a few quick changes here.

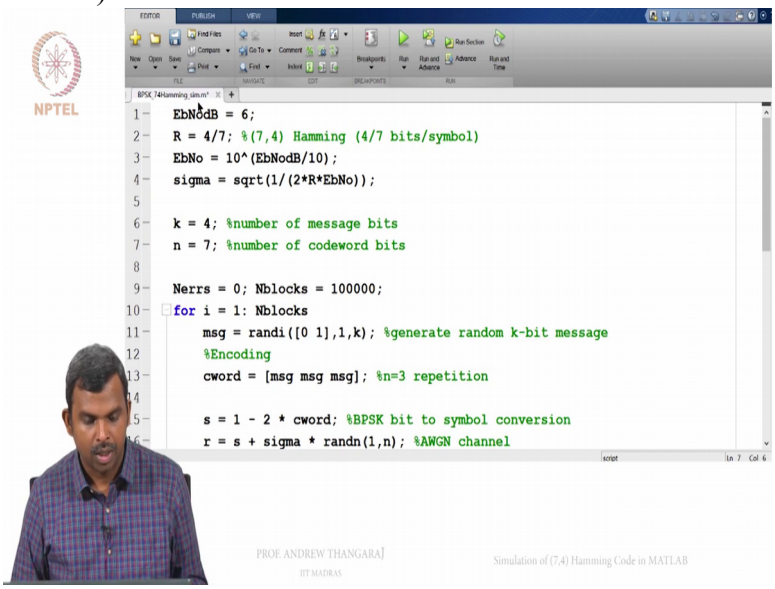
This part

(Refer Slide Time 03:41)



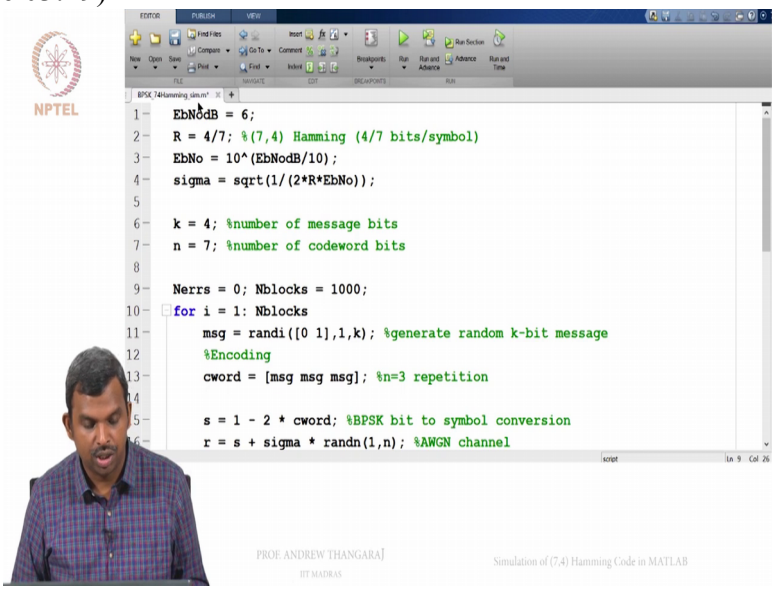

does not change, k equals 4, n equals 7.

(Refer Slide Time 03:45)




The number of blocks maybe we want to keep it at some low number

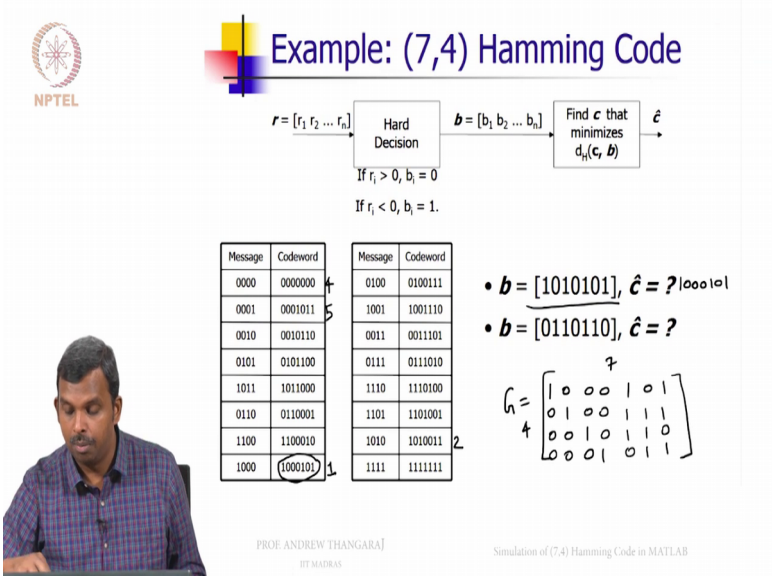
(Refer Slide Time 03:49)




just for simulation. This part is Ok. The message is k bits. Now the codeword I have to change, right.

So I have to have the message appearing by itself, Ok and then remember from my encoding, my

(Refer Slide Time 04:05)

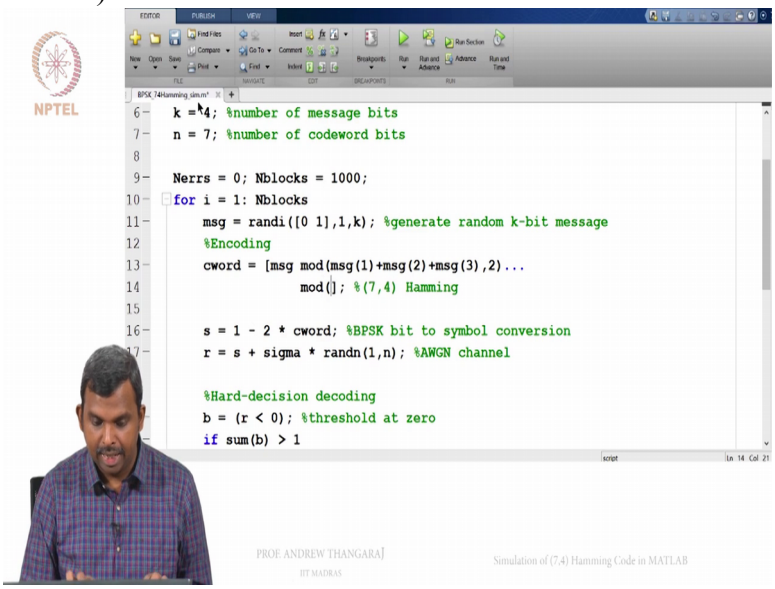



message will come here by itself. The 4 bits of the message will come here. What about the first parity? It is the first bit of the message XORed with the second bit of the message XORed with the third bit of the message. Ok

So that is my first parity, so I can write that down here. So it is going to be m s g of 1 plus m s g of 2 plus m s g of 3 and then I have to do modulo 2 and that you can do in this fashion, Ok. So this is for the 7 4, 7 4 Hamming, Ok.

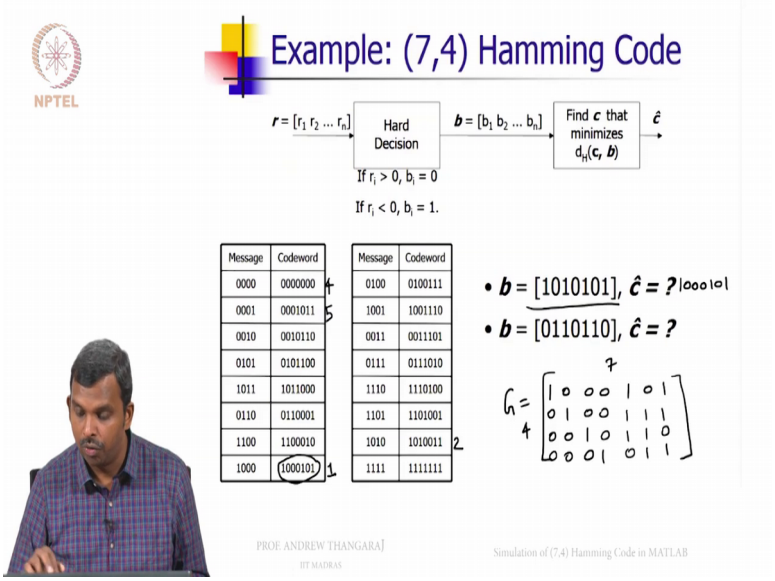
Now just to make the writing a bit clear I will use this dot dot dot which gives me the ability to write over multiple lines.

(Refer Slide Time 05:01)




The second message, if you, the second parity bit if you look at it

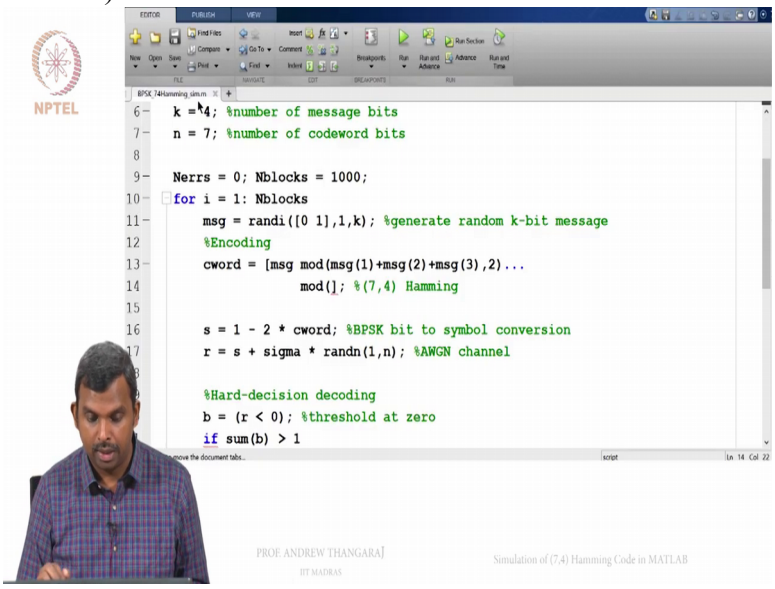
(Refer Slide Time 05:05)




is actually m, m the second message bit XORed with the third message bit XORed with the fourth message bit, right?

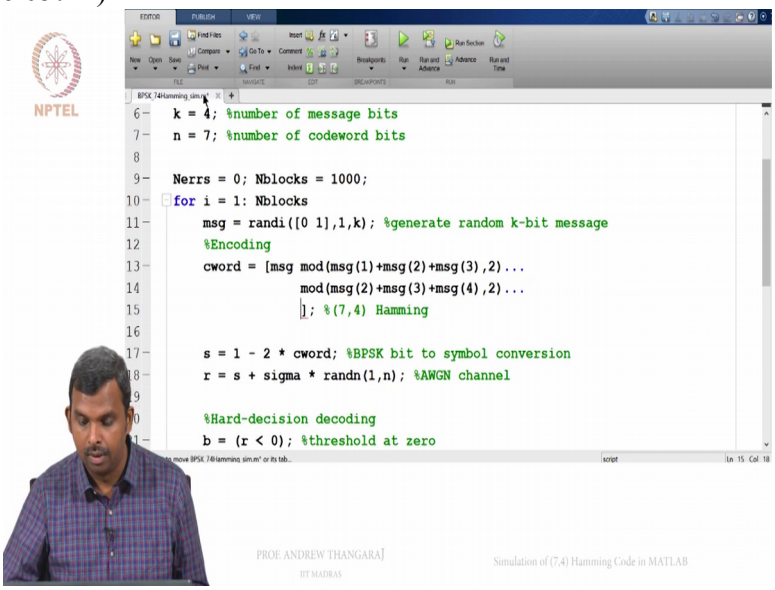
So I can

(Refer Slide Time 05:13)



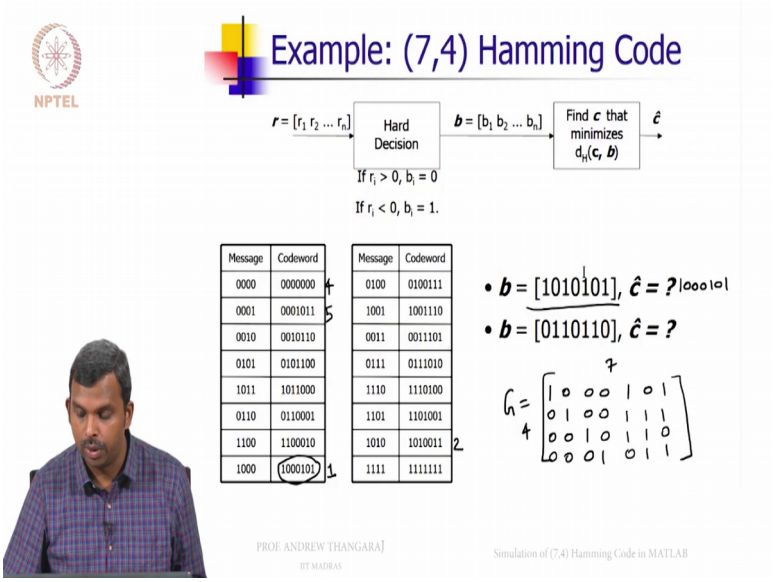

write that down. Remember this mod 2 you have to keep doing, Ok

(Refer Slide Time 05:24)




and the third parity bit, oops sorry, (()) the third parity bit if you look at the

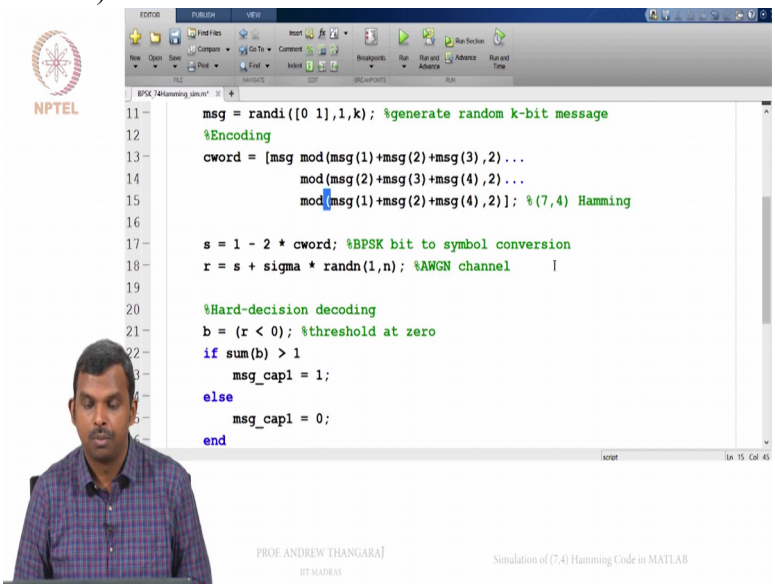
(Refer Slide Time 05:33)




equation once again, it is the first message bit XORed with the second message bit and XORed with the fourth message bit, Ok.

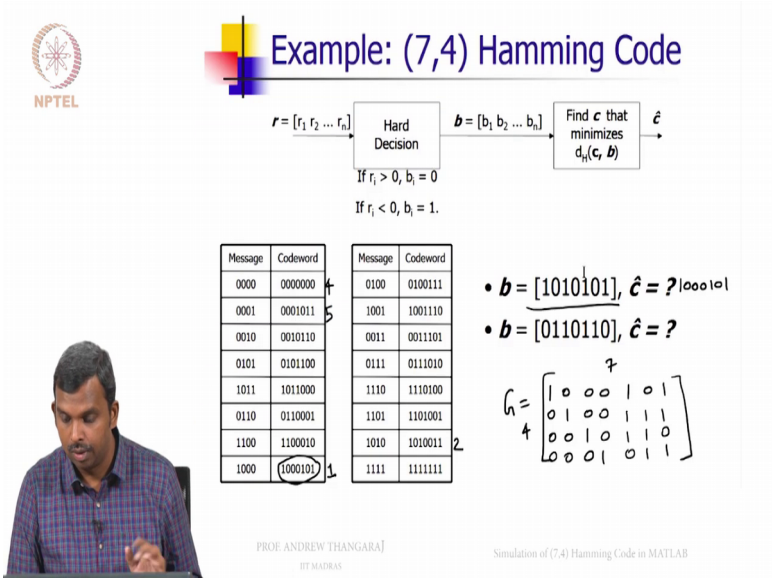
So one can write this down as m s g of 1 plus m s g of 2 plus m s g of 4 and comma 2 that is it.

(Refer Slide Time 05:51)



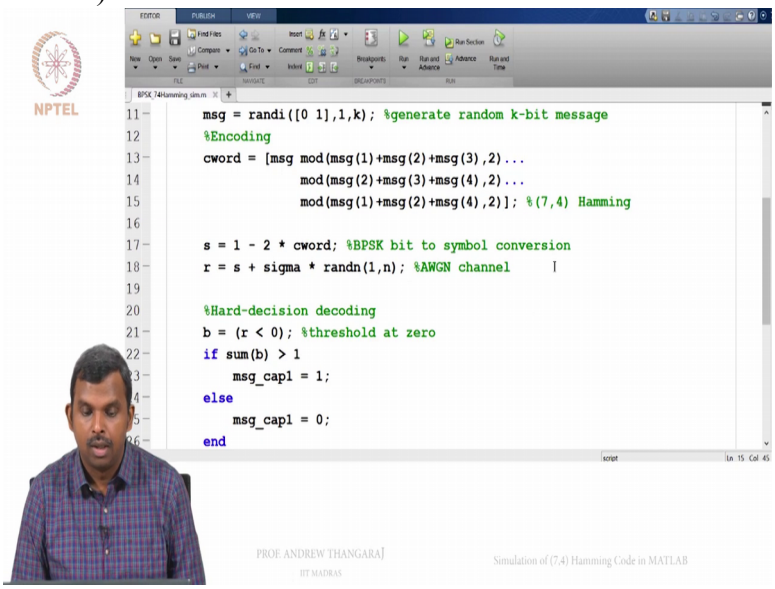

So I have done my encoding, Ok. So 1, 2, 3; 2, 3, 4 and 1, 2, 4, so let us check that once again,

(Refer Slide Time 06:01)




exactly what it is, Ok.

(Refer Slide Time 06:05)

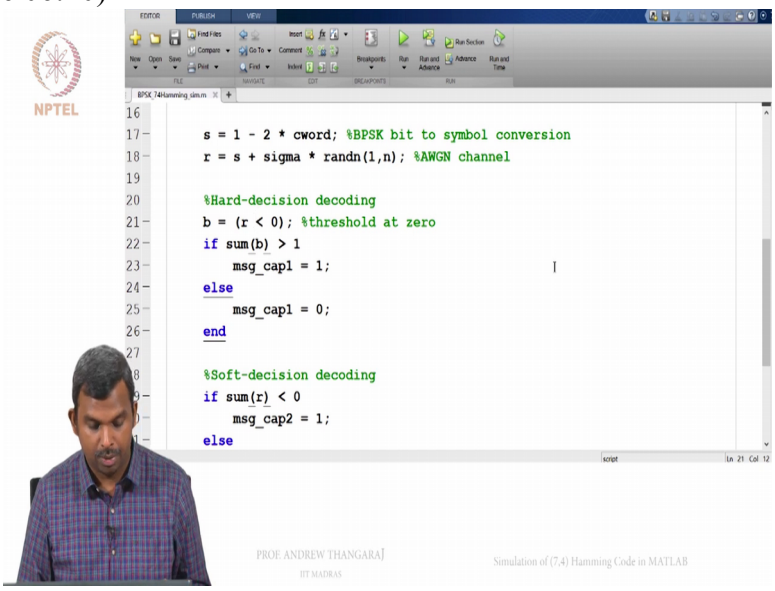



So that is done. Ok, you have done the encoding. You can write this a bit differently. You can store the generator matrix and do it if you like. There are various ways of doing it, Ok.

So after this the bit to symbol conversion is straightforward. It is the same thing. Nothing needs to be changed. This gives you the symbol vector. And then this gives you the received vector, right? I am adding, creating noise of Gaussian distribution multiplying with sigma and then adding it to this.

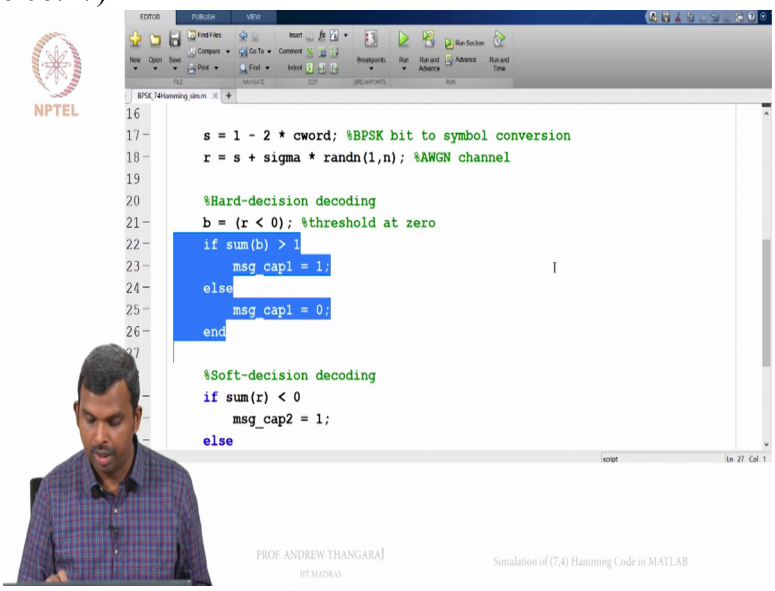
Now hard decision decoding, the threshold at zero is the same but this condition is not going to be the same,

(Refer Slide Time 06:40)




but this condition is not going to be the same, right? So need to have something different here; it cannot be the same as this. We will have to do something different

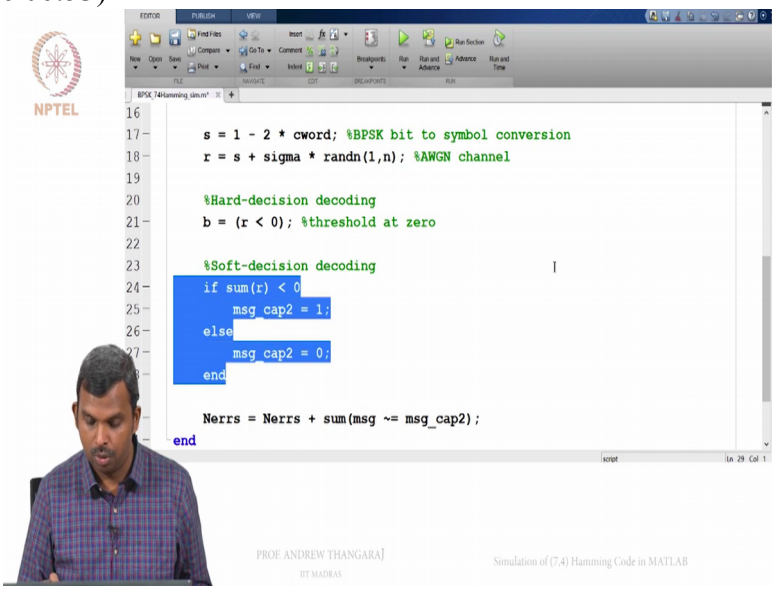
(Refer Slide Time 06:47)




here so let us delete this.

Same thing with soft decision decoding,

(Refer Slide Time 06:53)




this cannot be the same. This will have to be real out, Ok. Remember what we do in hard decision decoding, you have to find the distance of the hard decision vector b from every codeword and find that codeword which is closest to it, Ok. So that is the task in hard decision decoding.

And soft decision decoding, you have to find the distance between the received word, right, real received word and all possible symbol vectors, right and then find that symbol vector which is closest to, closest to the received vector in Euclidean distance, Ok.

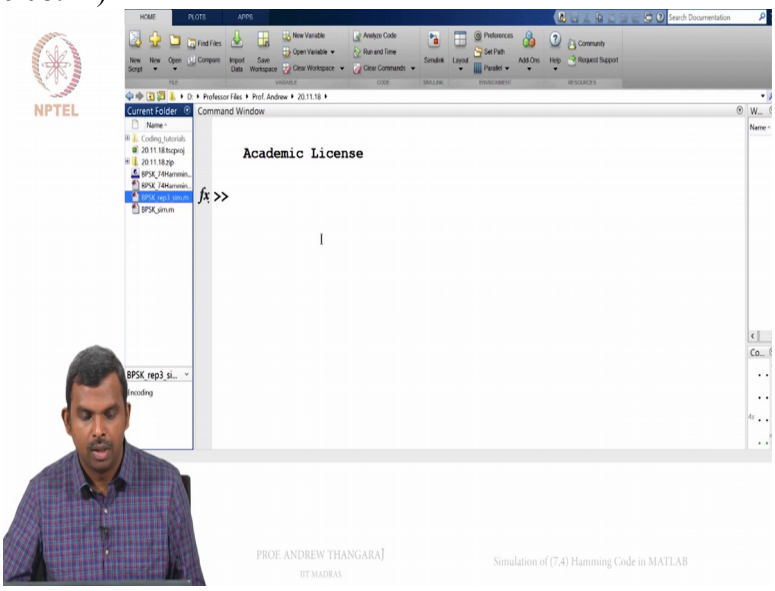
But we also saw B P S K K S instead of finding the Euclidean distance and finding the minimum symbol vector; you can take a dot product and find the maximum dot product. So which symbol vector has maximum correlation with your received vector?

So that is also something that you can find, Ok. So once we will do it in that fashion. We will do soft decision decoding in that fashion. I will take dot product with all symbol vectors and pick that symbol vector which gives me the best one.

So for both these things I need the list of codewords, Ok. So I need the list of all codewords of the Hamming code. And I want to store them in some, in a vector and then use it, Ok. And also all symbol vectors right. So right, so both of these, both of these I need. I am going to generate it.

For small codewords, codes like this one can quickly generate. It is not very difficult to write a piece of few lines of code to generate these codewords. But in general if you have larger code, it might be difficult and it won't work, Ok. But nevertheless let us get started with the, with generating these codewords

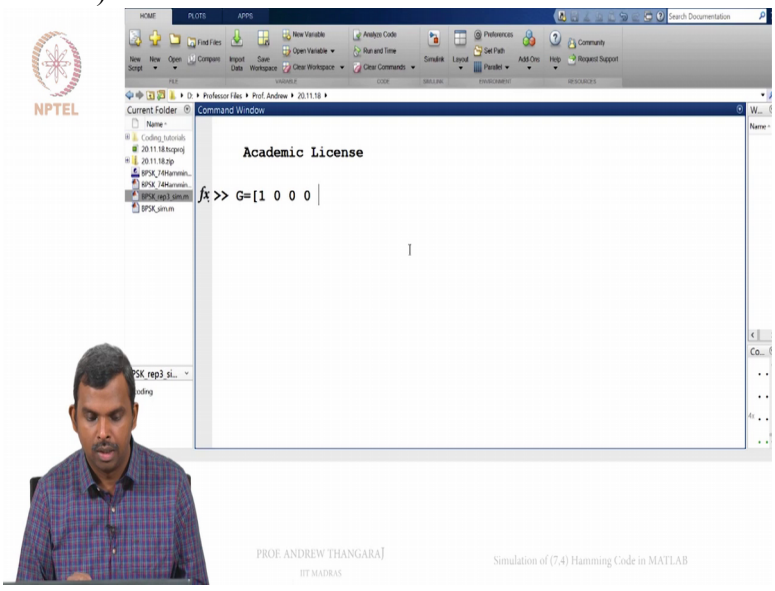
(Refer Slide Time 08:41)




Ok.

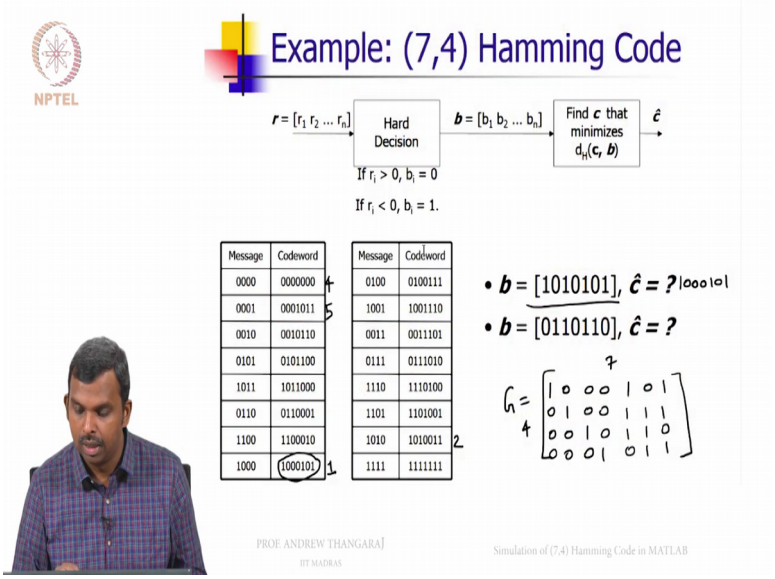
So I will do, I will describe how I usually generate codewords. This is not the only way to do it. There are so many other ways in which one can do this so one can write down the, first generator matrix, Ok.

(Refer Slide Time 08:56)



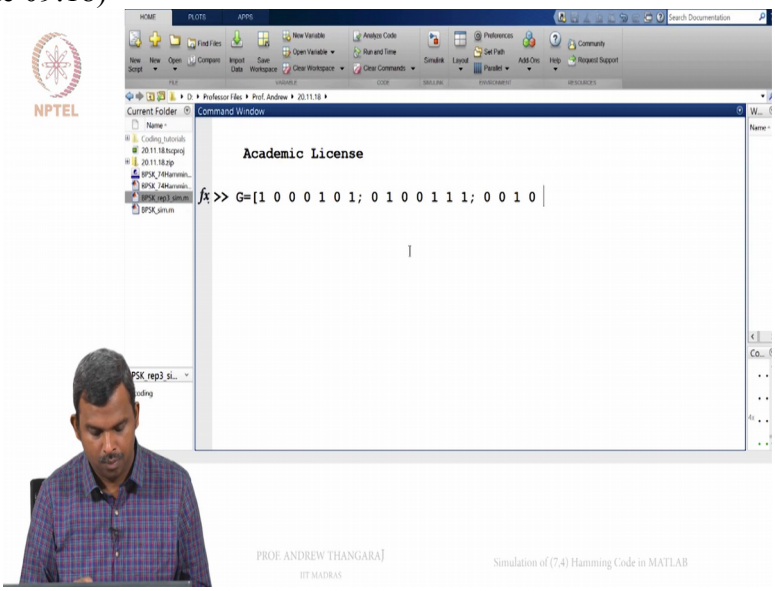

So let me take some help from what I have before,

(Refer Slide Time 08:59)



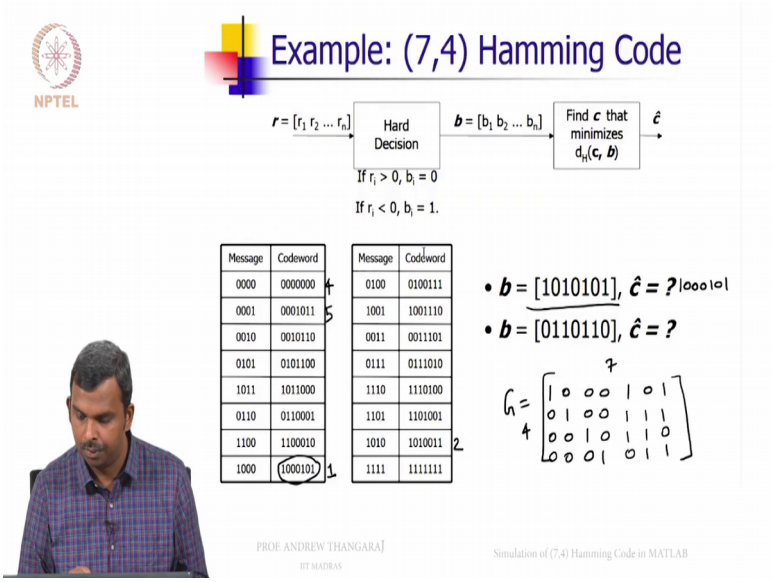

1 0 1, 1 1 1 (()) Ok.

(Refer Slide Time 09:18)



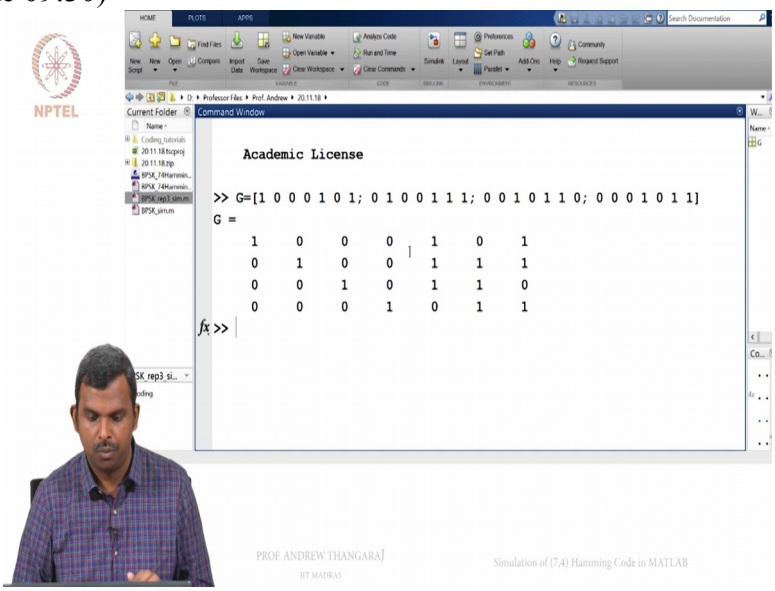

Let us check.

(Refer Slide Time 09:19)



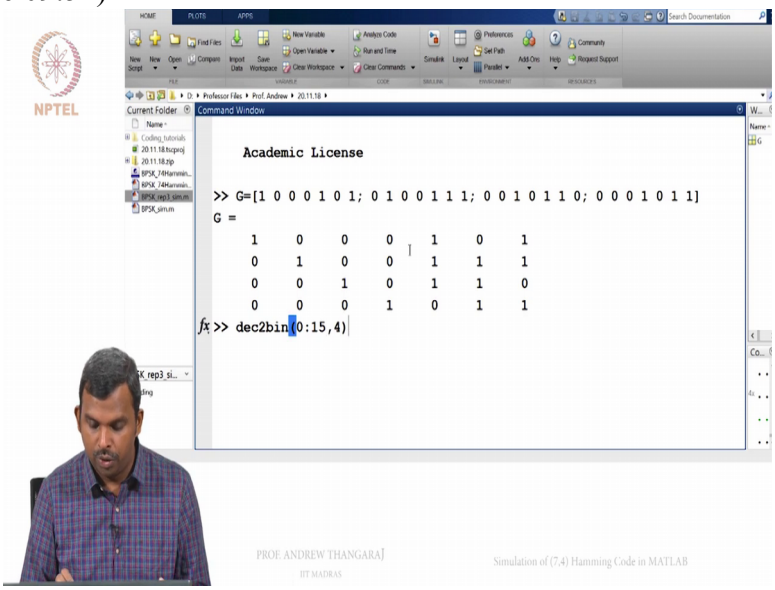

1 0 0 1 1 ((0)) , Ok

(Refer Slide Time 09:30)



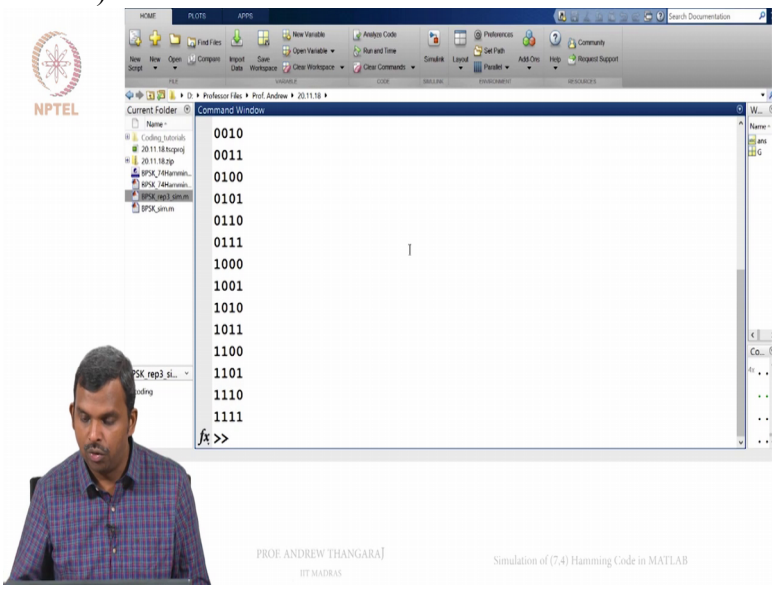

so you can check. This is the generator matrix that I had before, alright. So now I have to multiply with the message bits. How many message bits do I have? I have 4 message bits so there are 16 possible binary vectors. So how do you get a list of 16 possible binary vectors? So it turns out that there is this little command here which I use. There are, I am sure other possibilities here.

(Refer Slide Time 09:54)



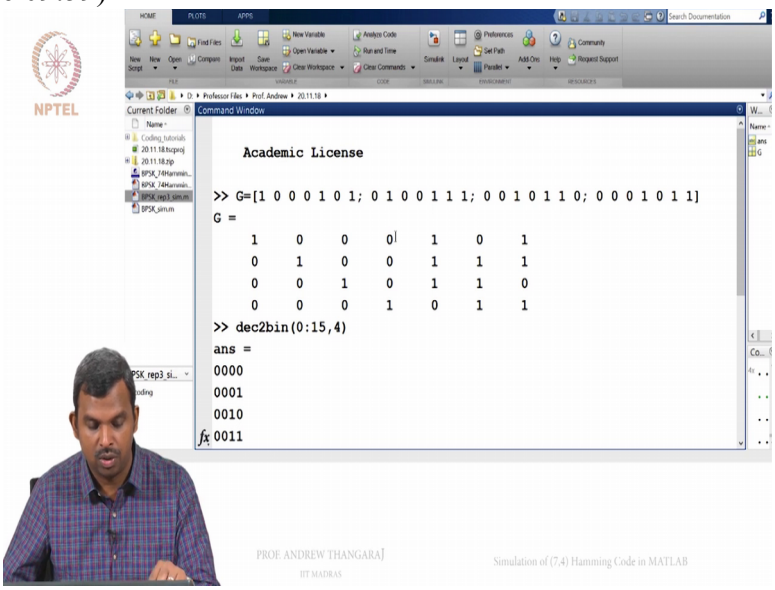

So if we do this,

(Refer Slide Time 09:55)




it will give you the list of all binary vectors in Matlab. Ok, so

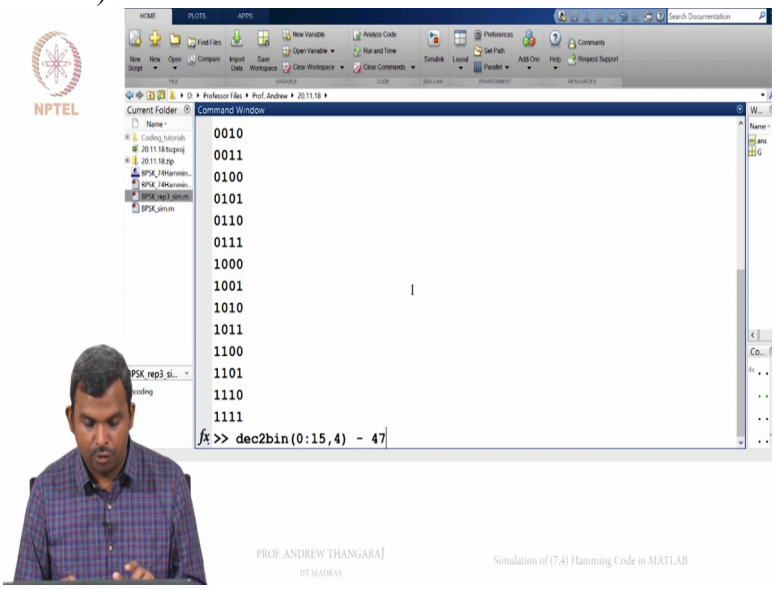
(Refer Slide Time 09:59)




0 0 0 0, 0 0 0 1, 0 0 1 0. So the problem is this is a string and it turns out, I want to convert it into vectors, right?

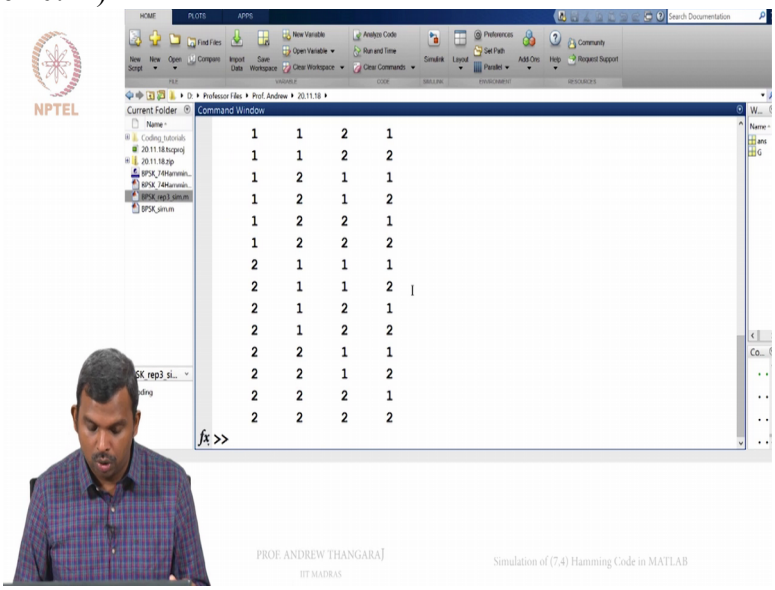
I do not want a string, I want a matrix. For that I use this little trick. I usually subtract it,

(Refer Slide Time 10:13)



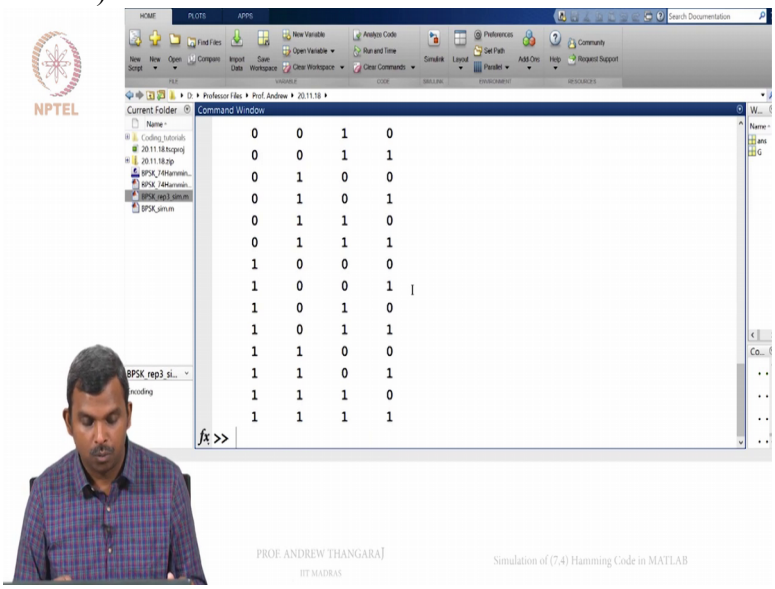

I think 47

(Refer Slide Time 10:14)



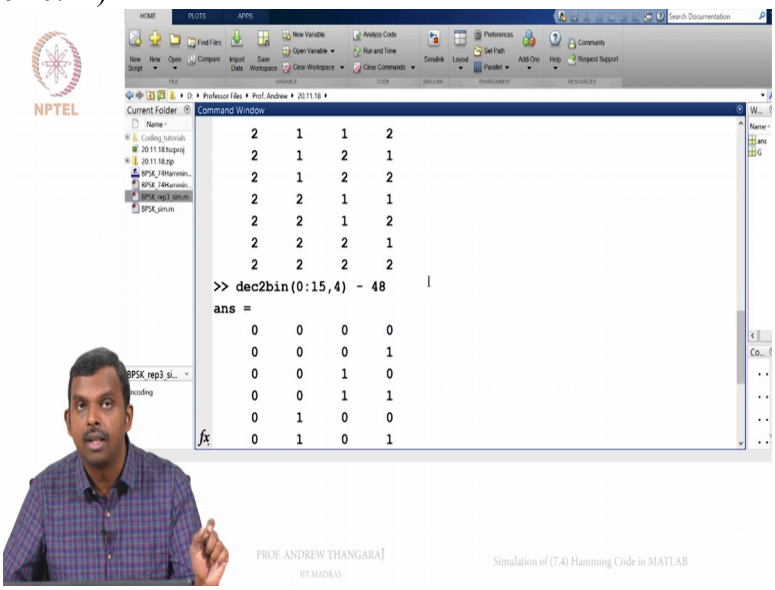

will do it, no it will not. 48 would do it.

(Refer Slide Time 10:18)




Ok. So if you do d e c to b I n

(Refer Slide Time 10:21)

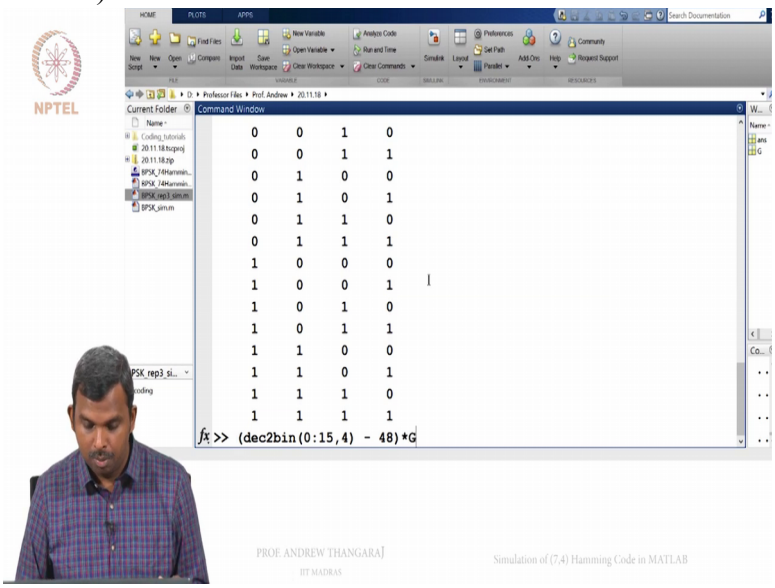



and then you subtract 48 from it, Matlab does this conversion from strings to bits at least.

So you have 0 1, this is a good enough trick to use so I will do this. So this gives me the vector of all possible binary sequences of length 4. Ok, you might have other ways of generating this. So I am just generating it like this, Ok, 0 0 0 0, 0 0 0 1 so on till 1 1 1 1.

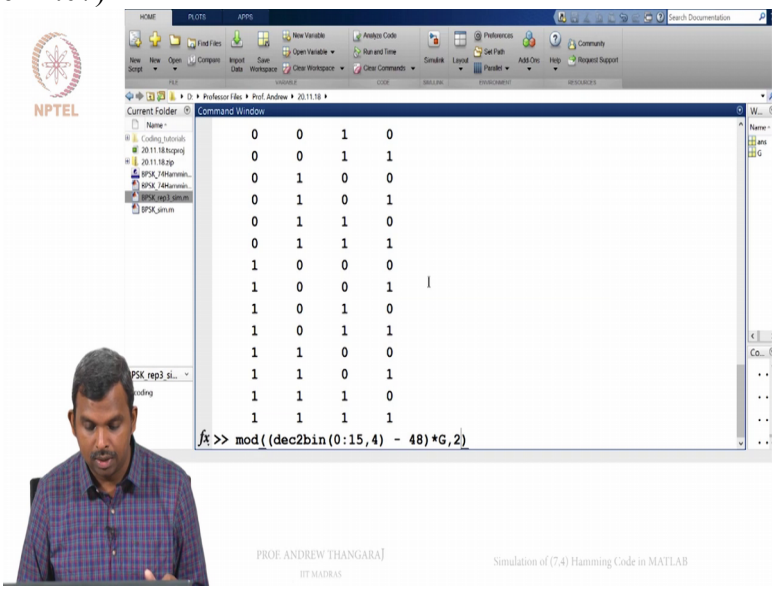
Ok this is 16 vectors here. So let us store this, you don't have to necessarily store this, you can just keep this and multiply with G on the right. Ok

(Refer Slide Time 10:59)




but we have to do modulo 2. If you do not do modulo 2, you won't get right answers. Ok, mod of this comma 2.

(Refer Slide Time 11:07)

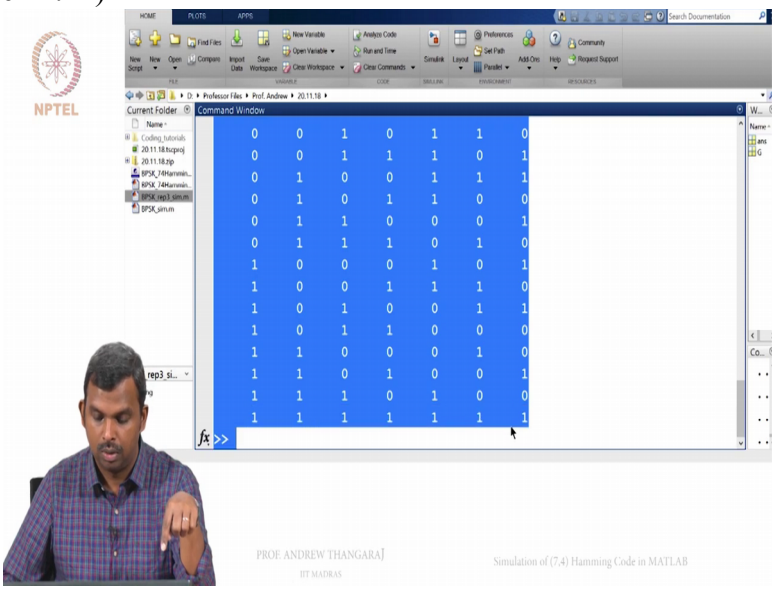



So turns out this is the list of all codewords that I want, Ok.

So this is the list of all codewords in the Hamming code. You see I found all the binary sequences of length 4, multiplied with G. I did a modulo 2. I got the, I got all codewords, Ok. So you can check if you like that we got the codewords, Ok.

So now I can do a simple cut and paste from here to my editor window,

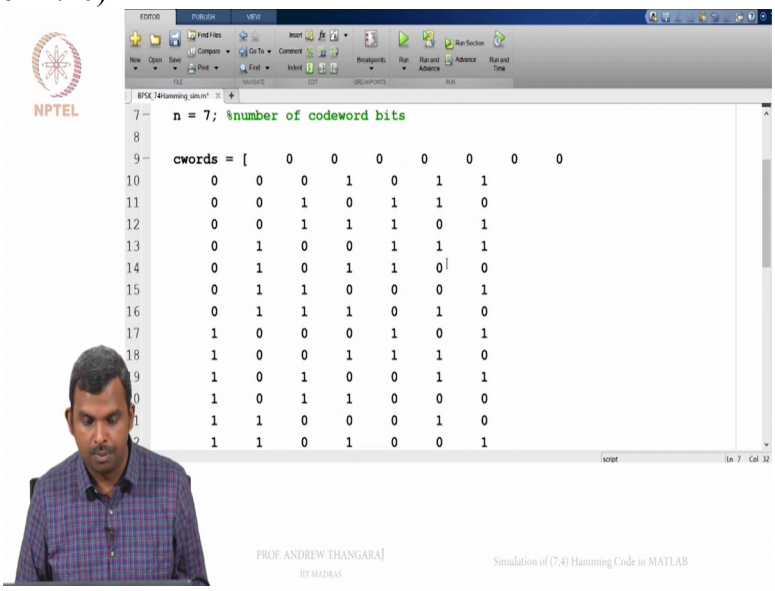
(Refer Slide Time 11:44)




Ok. So one can, I mean one need not do it inside the simulation block. You can do it outside. I can call it Ok, codewords Ok or you can use any other notation that you like. So here I can paste it.

So it is good to put

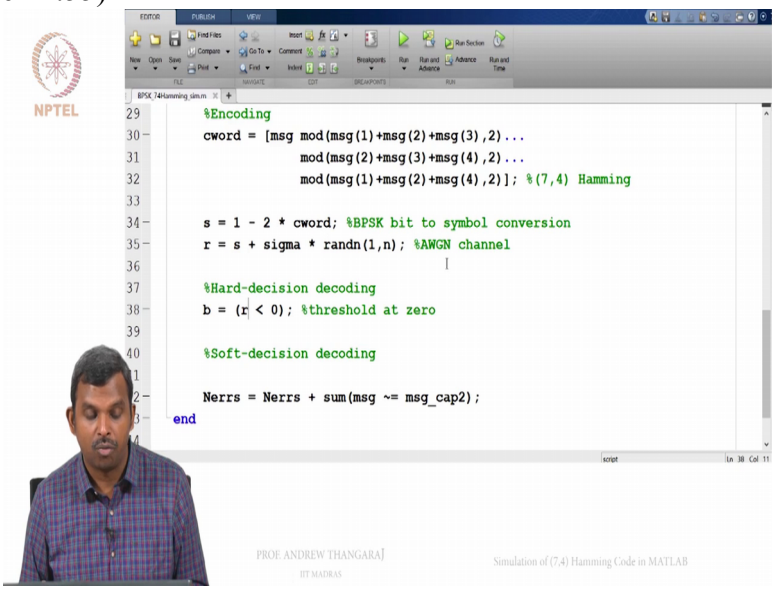
(Refer Slide Time 12:20)




semicolons to indicate that the...So I want to immediately point out, this is something that you cannot do for long codes, Ok. So if you have, if you have say k equals 100 or 1000, this will be 2 power 100, you cannot list out all the codewords like this but for us it is Ok.

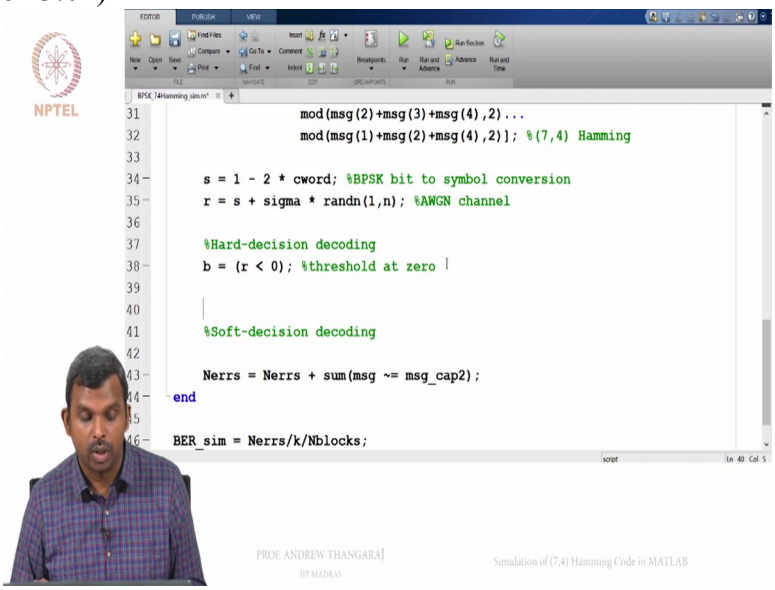
So this is a small enough operation. You can do it very easily, Ok. So once I have the list of codewords

(Refer Slide Time 12:55)




and I have my hard decision vector b, it is relatively easy to write down the hard decision decoding, Ok.

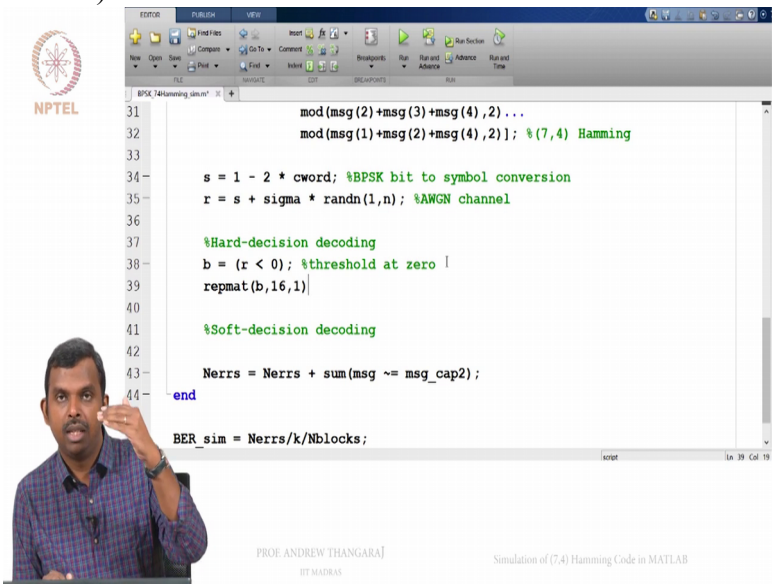
(Refer Slide Time 13:04)




So I have to find the distance between this vector b and every codeword in my array c words, Ok. So I had the lost of all codewords. I have to find the distance between this and that, Ok. So even for this I am going to use a little bit of a device here. It is just to get it done very quickly.

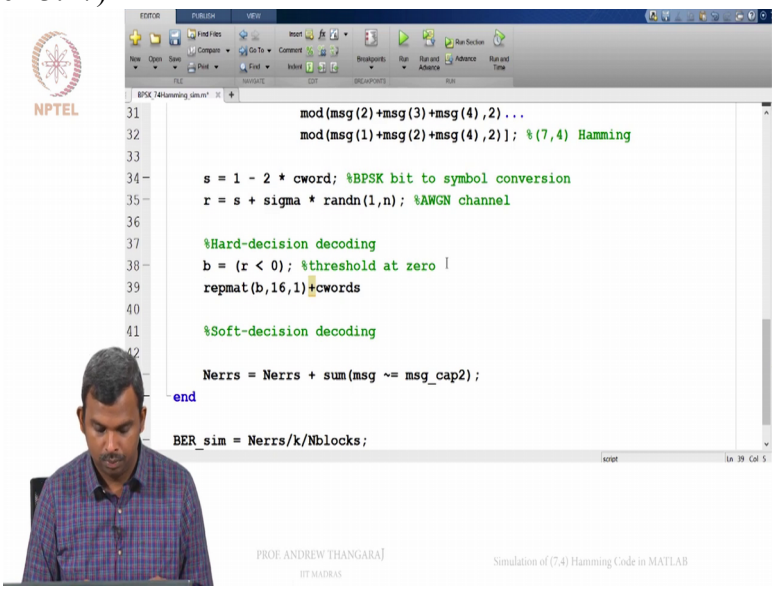
So I can do this repeating the matrix, Ok. So I will do it 16 times 1, Ok. So this will create a vector of b

(Refer Slide Time 13:35)

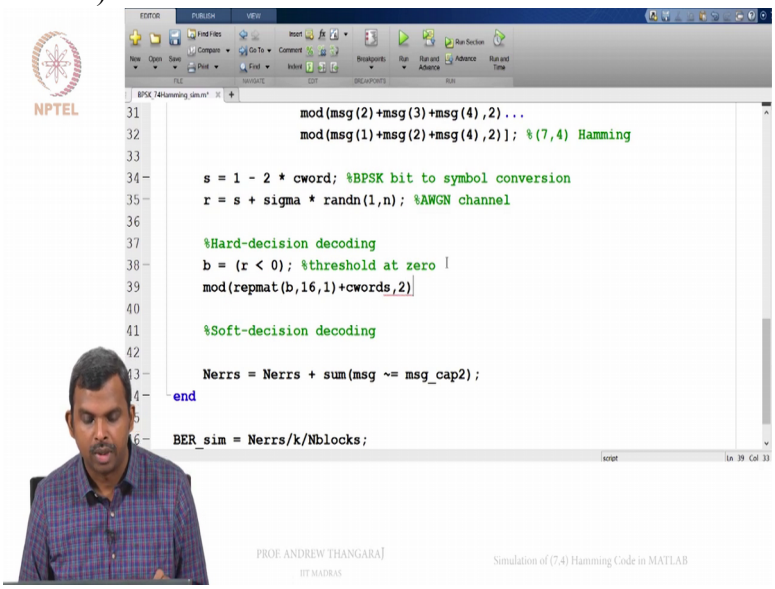



repeated 16 times. So you can try this out if you have installation, Ok. And then I will add it to c words and do a modulo 2,

(Refer Slide Time 13:47)



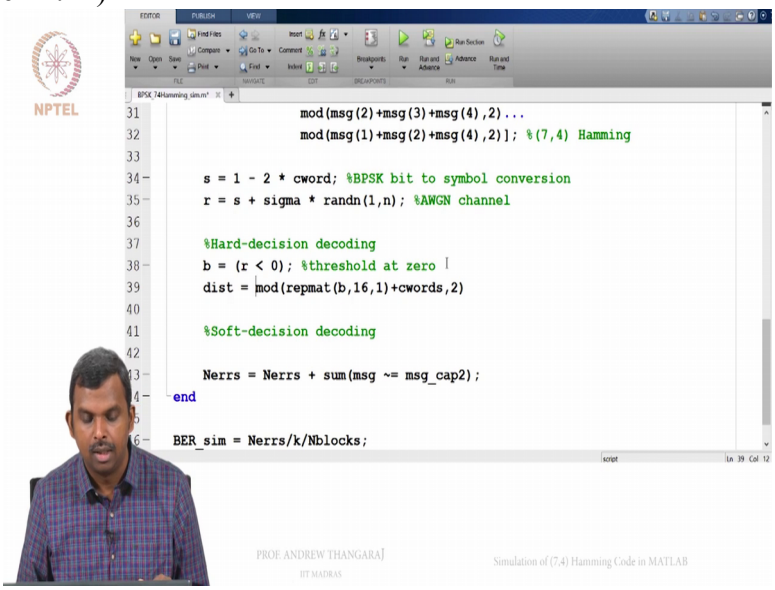

(Refer Slide Time 13:52)




Ok this will give me the XORs; I am XORing b with every possible codeword, Ok.

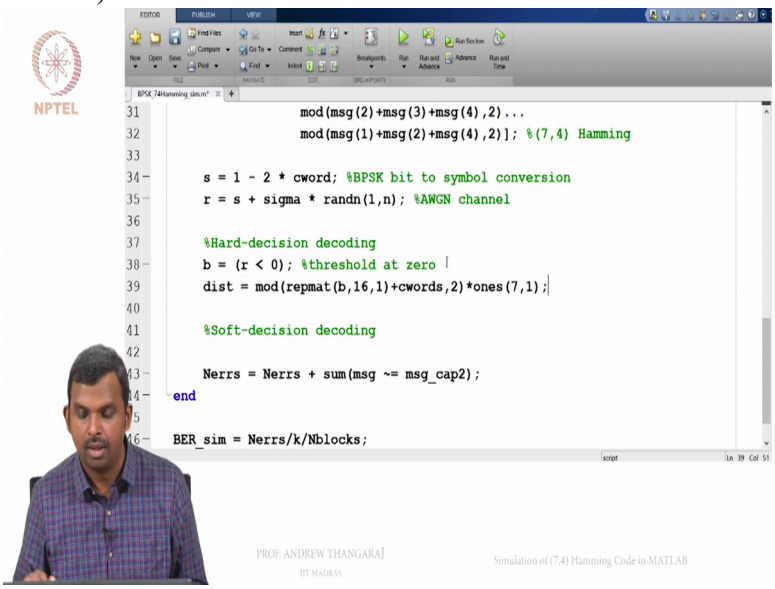
And then what do I do, Ok so this gives me just the XORs of all the codewords and now I can look at the weights, right? So, so this I will call as the distance array, Ok.

(Refer Slide Time 14:14)




So this is just the XORs, so how do I find the weight? So it turns out you can multiply a vector on the right, Ok and this will give me the distances,

(Refer Slide Time 14:32)



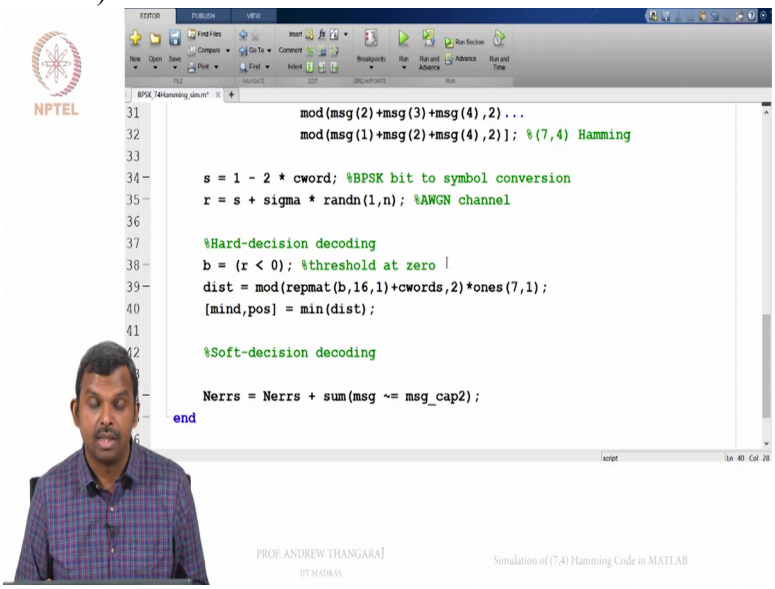

Ok.

So I will illustrate it may be later on we will see how this works out. So I am repeating the received vector b 16 times, I am doing XOR with each codeword. I get a list of XORs, they are 16 in number each is a vector of length 7. How do I find the weight of that now?

I multiply on the right with 1s of 7 comma 1. So it simply multiplies and adds up and gives me the weight, Ok. So I am not done now. I have to find, out of all these guys the one which has the least distance, Ok.

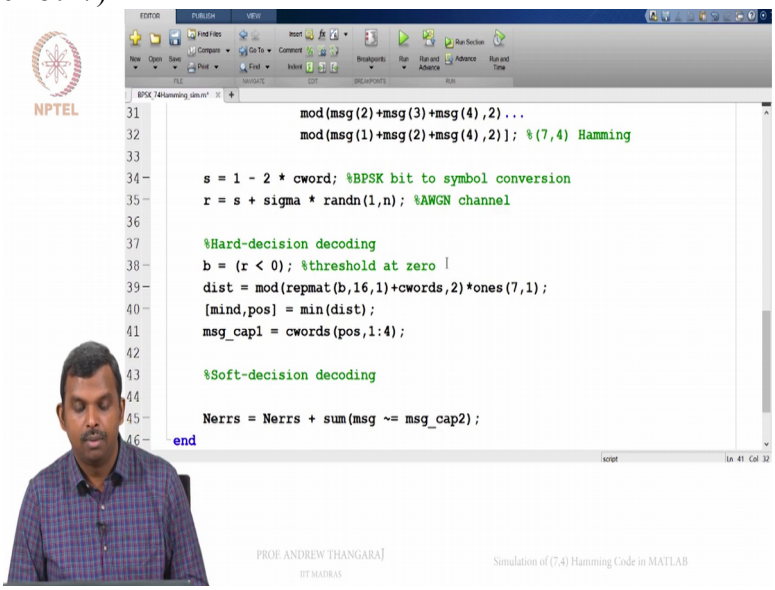
So that one can find using this min, so min d comma position equals min of distance. So of all these distances which one

(Refer Slide Time 15:24)




is the minimum, Ok? So that this solves and then once I know p o s, the minimum itself does not really matter. I can find message cap 1 as c words of pos comma 1 colon 4,

(Refer Slide Time 15:47)

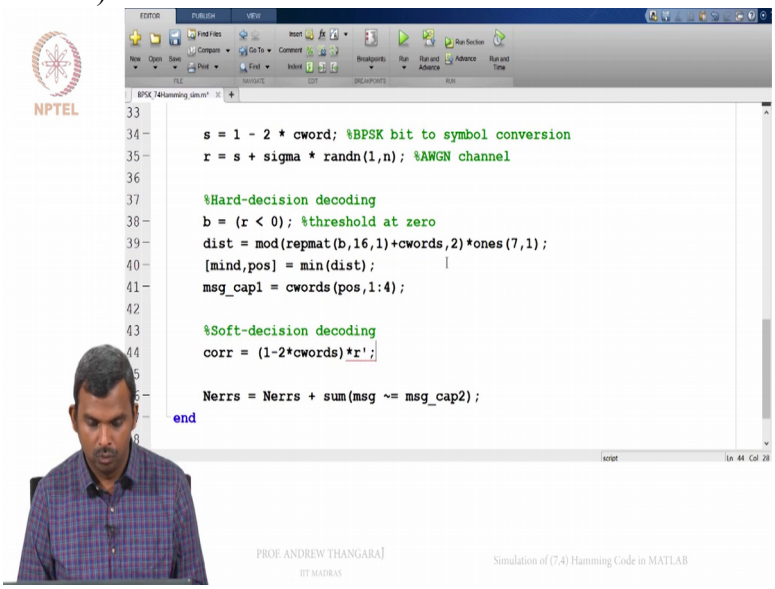



Ok.

So hopefully it was clear to people. I wrote it down a bit quickly. Think about what I am doing. I am finding the distance between b and every possible codeword and finding that codeword which is closest minimum distance between this and that, Ok. So let us do, so we will run this. I mean I have to check if I have made any mistakes or not. Maybe I made a mistake.

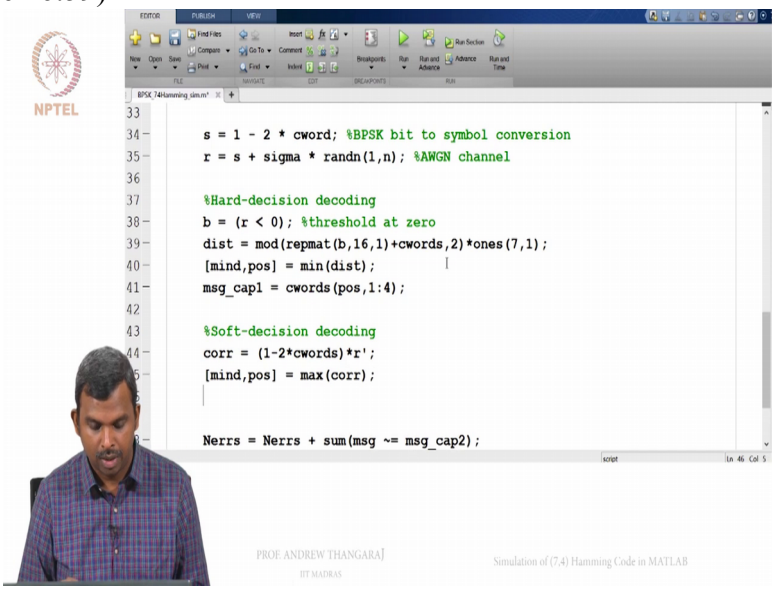
When we ran it we will know if it causes any errors, we will go and look at it and check it out, we will do that at that time. So soft decision decoding, it is similar except that the distance is going to be, it is not distance, I will call it correlation, correlation is going to be with the received word r but I am going to have the list of symbol vectors. So it is 1 minus 2 into c words multiplied by the transpose of r.

(Refer Slide Time 16:47)



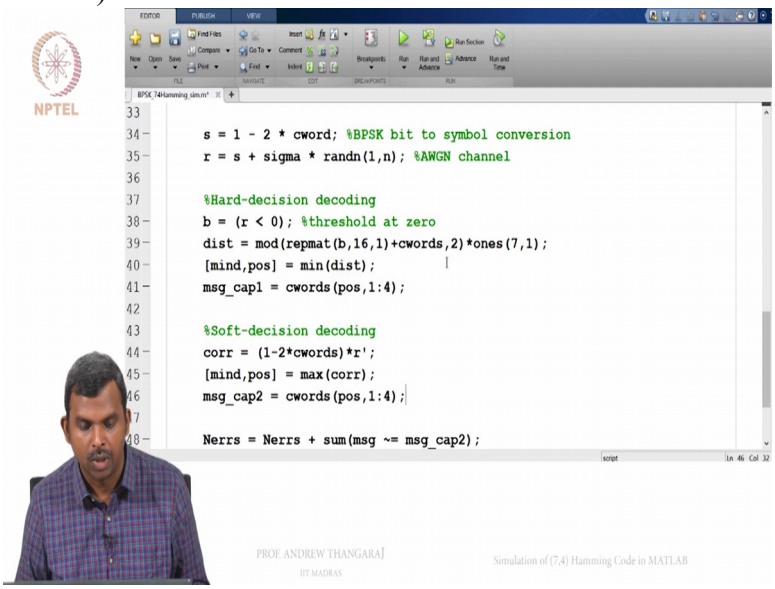

Ok, so this will give me the list of all correlations and once again find m min and pos but in this case, I have to do max, max of correlation

(Refer Slide Time 16:59)



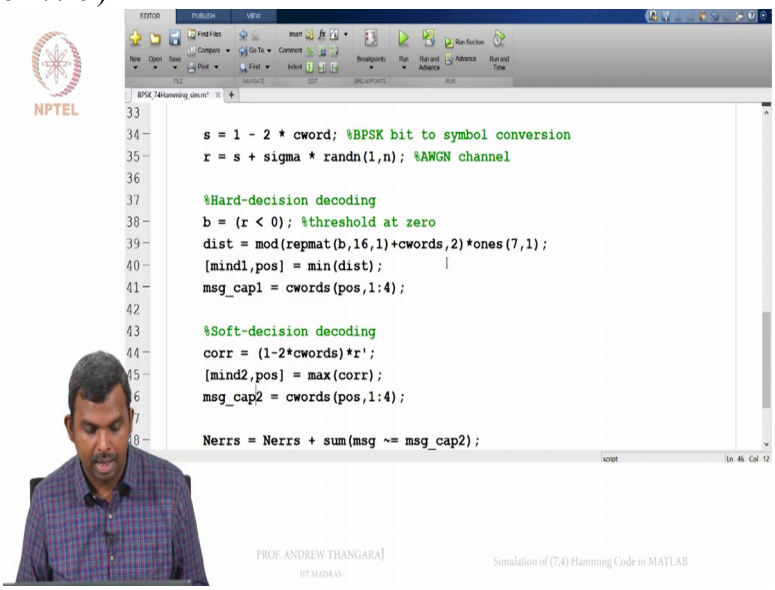

and then what I have here is the same,

(Refer Slide Time 17:08)




Ok. So, so Matlab is pointing out that, Ok so maybe I will call m ind1, m ind2 just to keep

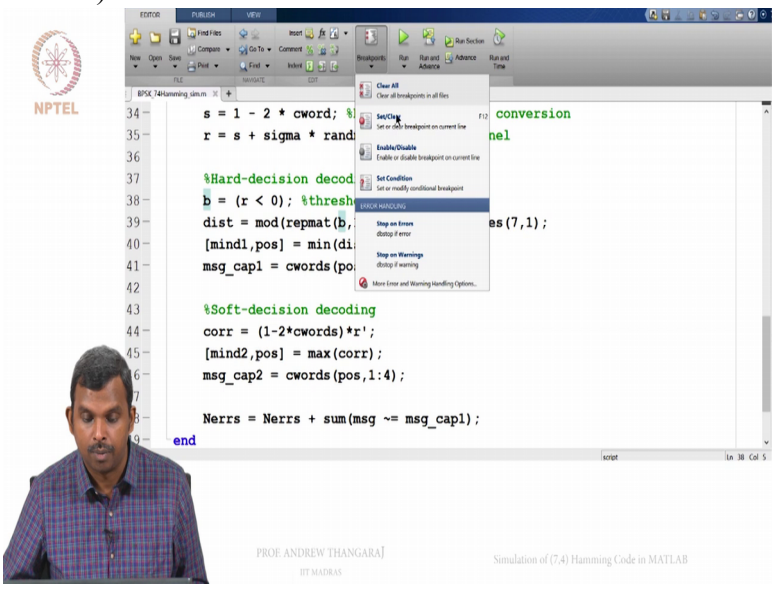
(Refer Slide Time 17:19)




Matlab happy and then this is just a, just a way of counting errors. May be I will use hard decision decoder first, Ok.

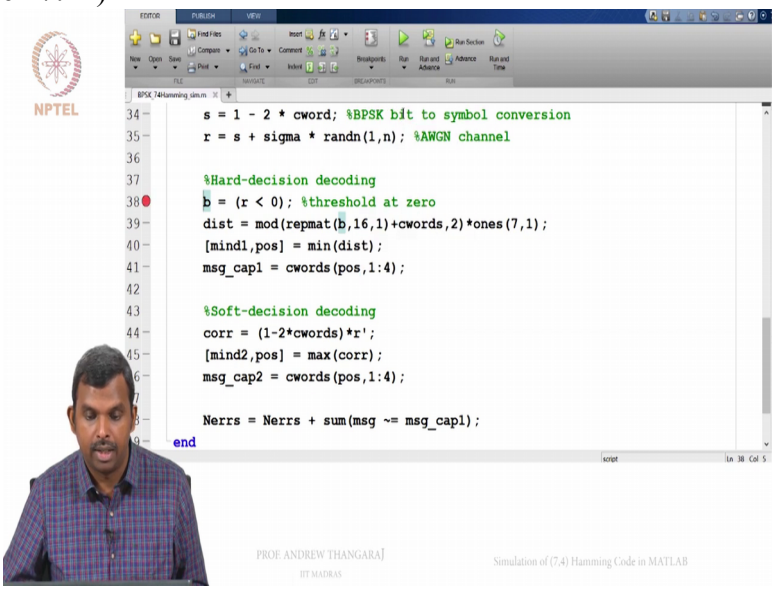
So let us see how this works. So one useful thing I do usually when I run this is I put a breakpoint here

(Refer Slide Time 17:40)



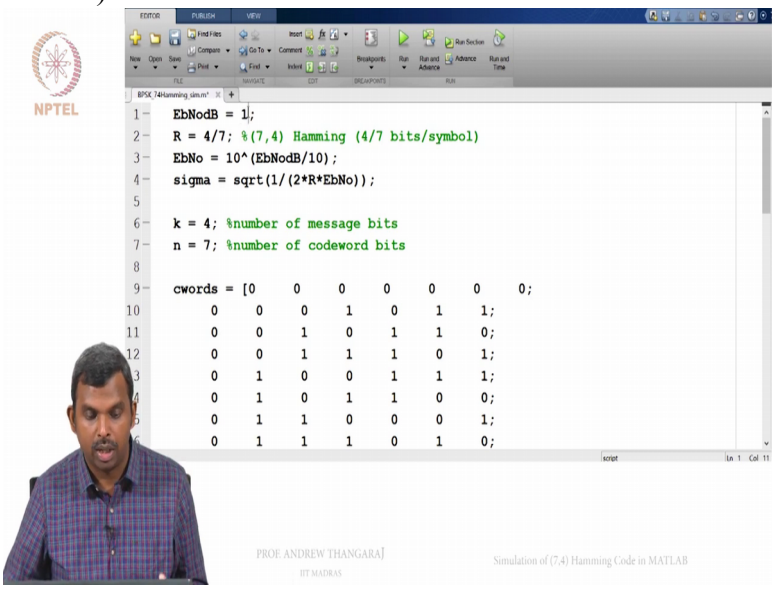

and then

(Refer Slide Time 17:41)



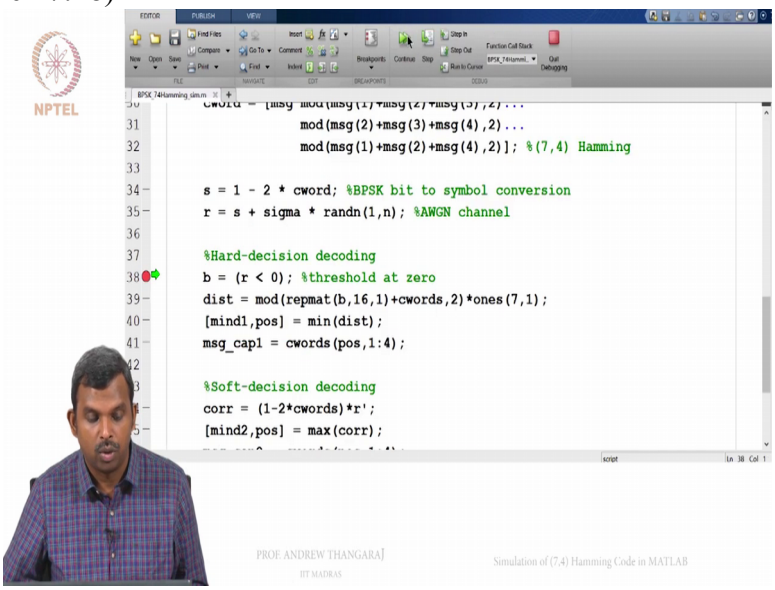

I run it to see what is going on. So maybe, maybe we will put lower

(Refer Slide Time 17:46)




E b over N naught and then let us just

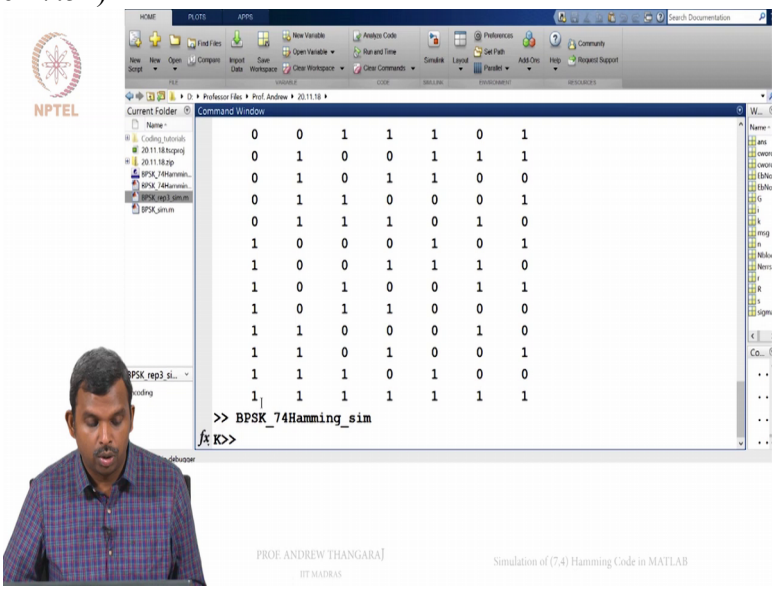
(Refer Slide Time 17:48)




run it, Ok.

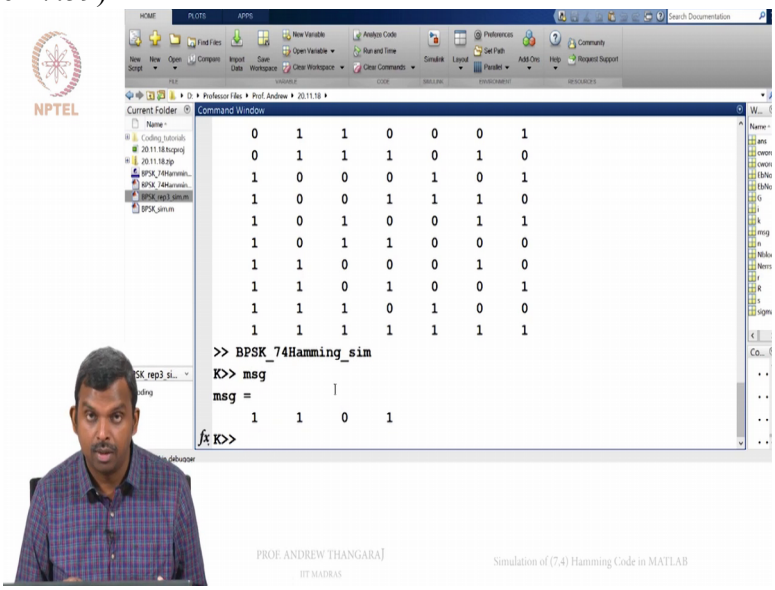
So it has come up to this point and stopped and to see what it is doing, you can go

(Refer Slide Time 17:54)



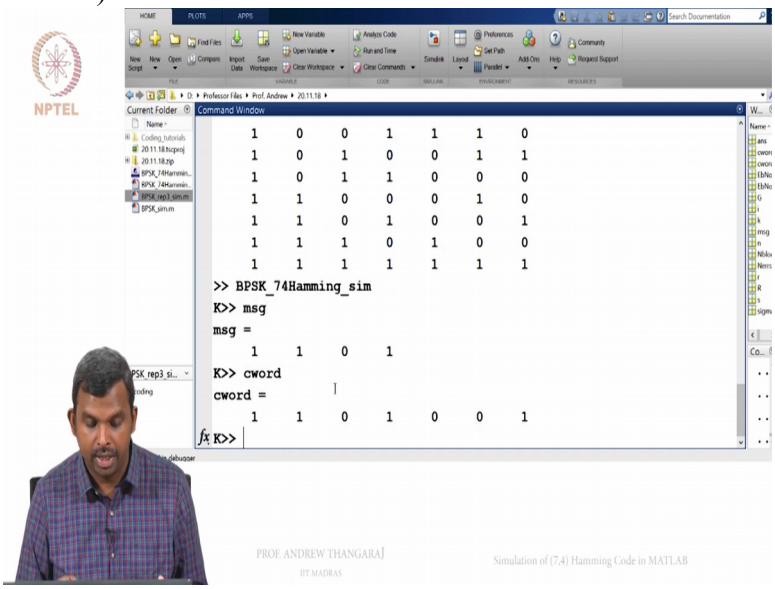

here and look at the message that was generated,

(Refer Slide Time 17:59)



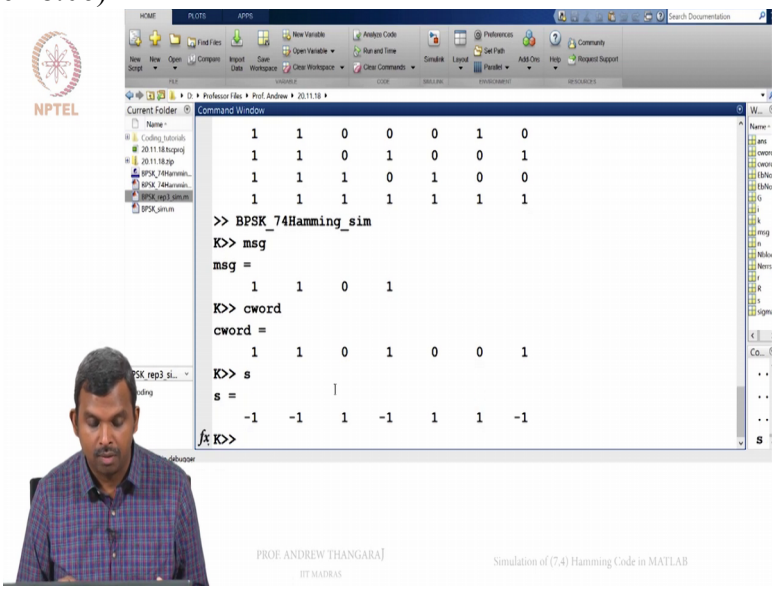

1 1 0 1, the c word

(Refer Slide Time 18:01)



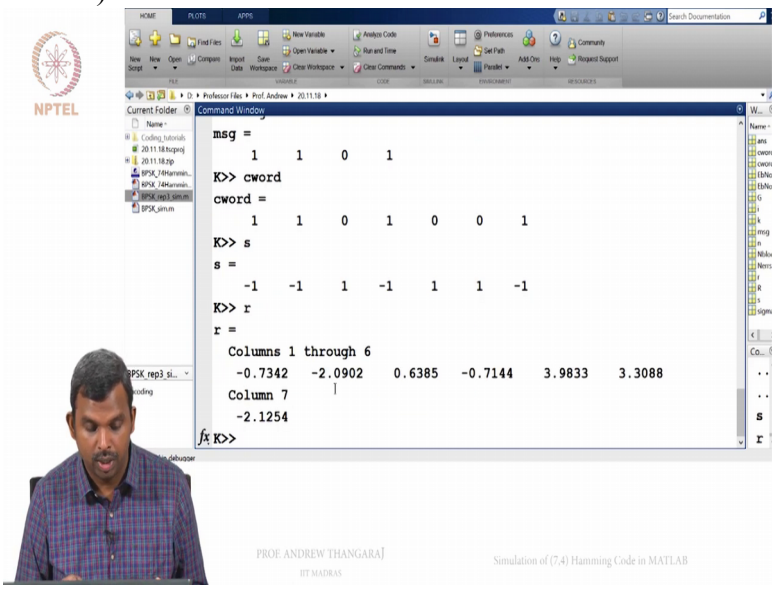

that was generated 1 1 0 1 0 0 1 and then s

(Refer Slide Time 18:06)




which is minus 1 minus 1 1 etc and then r,

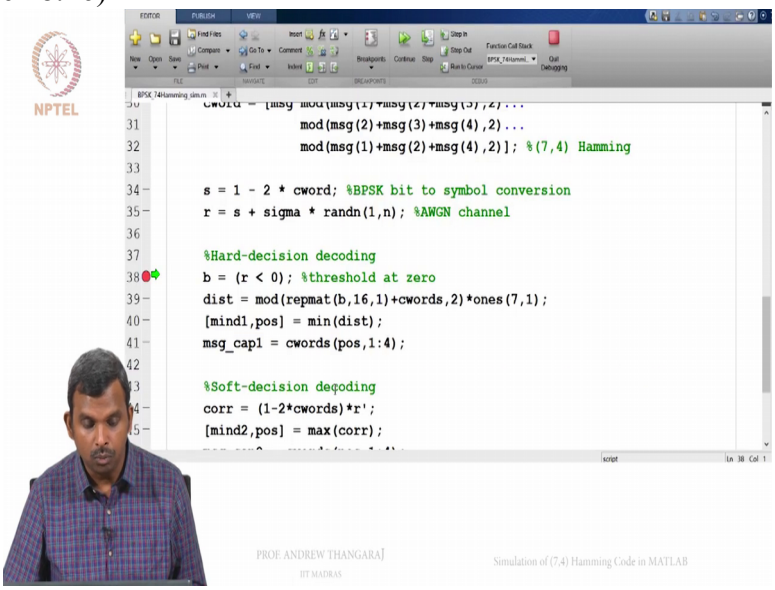
(Refer Slide Time 18:09)




Ok.

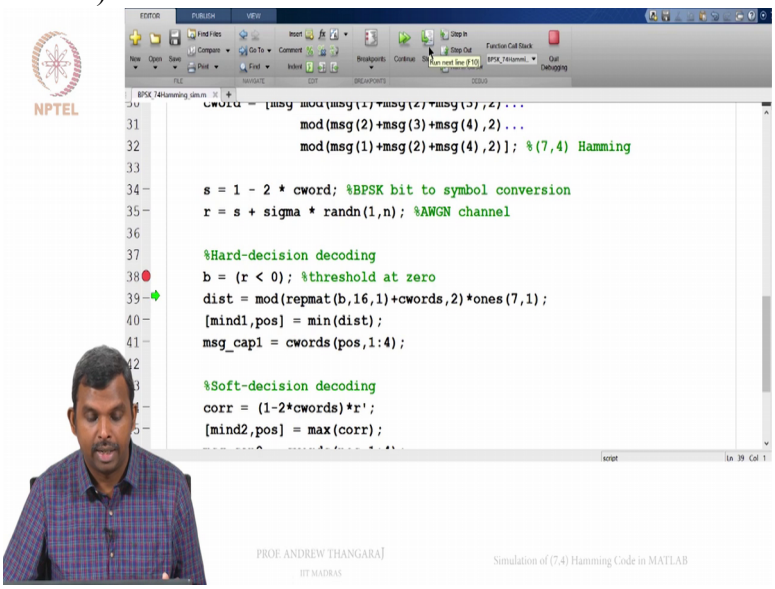
So you can see noise got added. Lot of things have happened. It does not appear there has been an error but nevertheless let us see what happens, Ok. So this is r,

(Refer Slide Time 18:26)



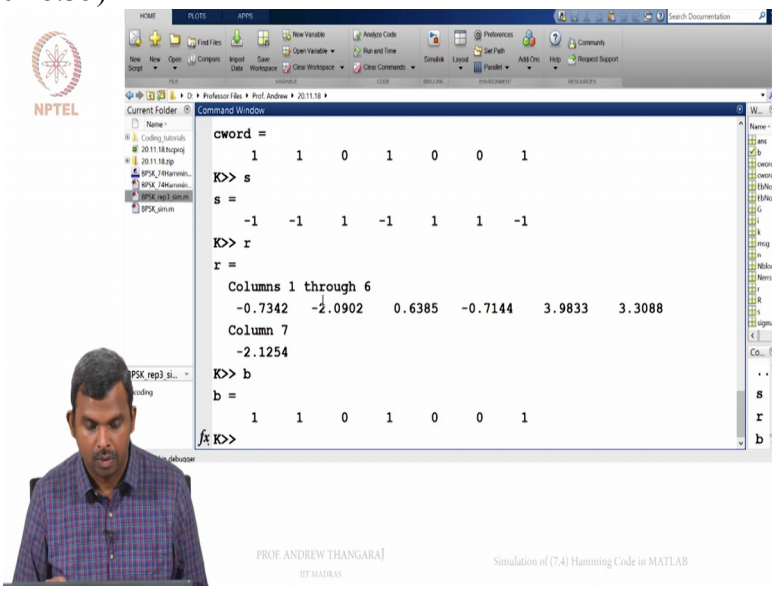

Ok. So I am going to run one step, so I will get

(Refer Slide Time 18:31)




b, Ok. So let us check what b is,

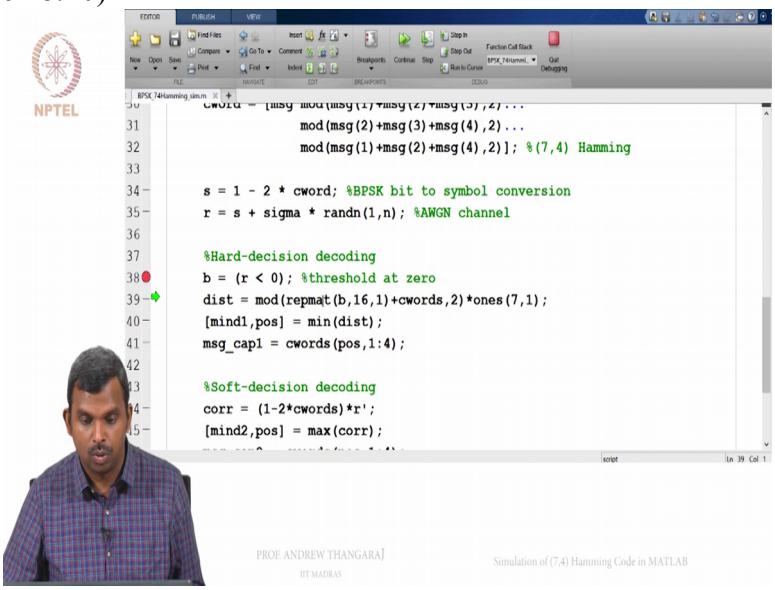
(Refer Slide Time 18:35)




Ok.

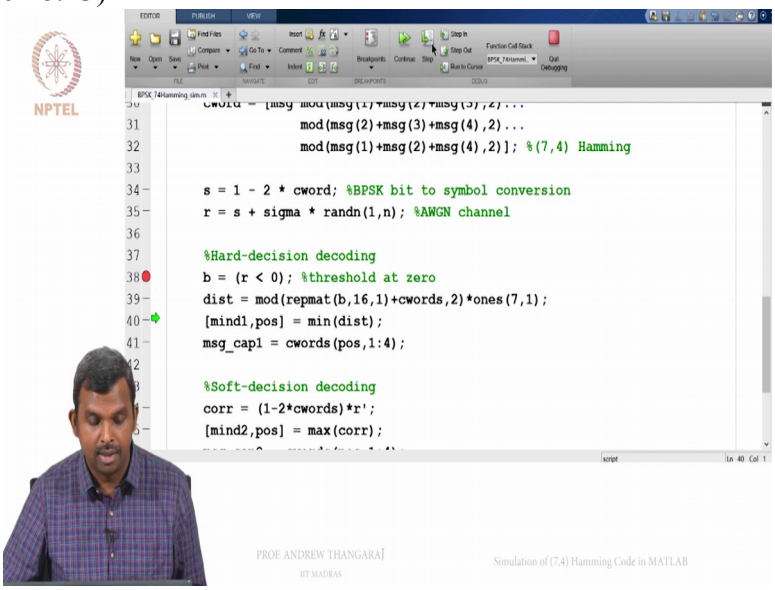
So it sort of agrees with the codeword here but nevertheless let us to go through

(Refer Slide Time 18:40)



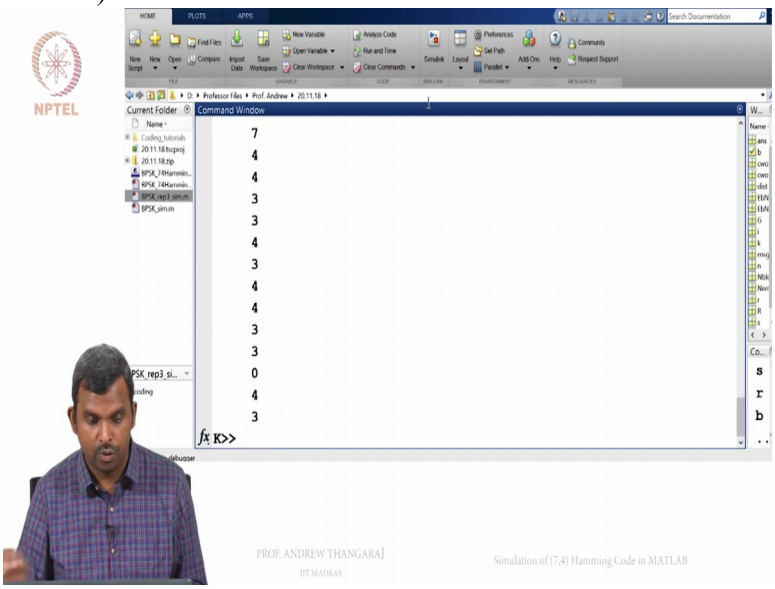

what the decoder is going to do. It is going to find distance,

(Refer Slide Time 18:43)




Ok. So let me show you how this worked. It gave you a array of distance.

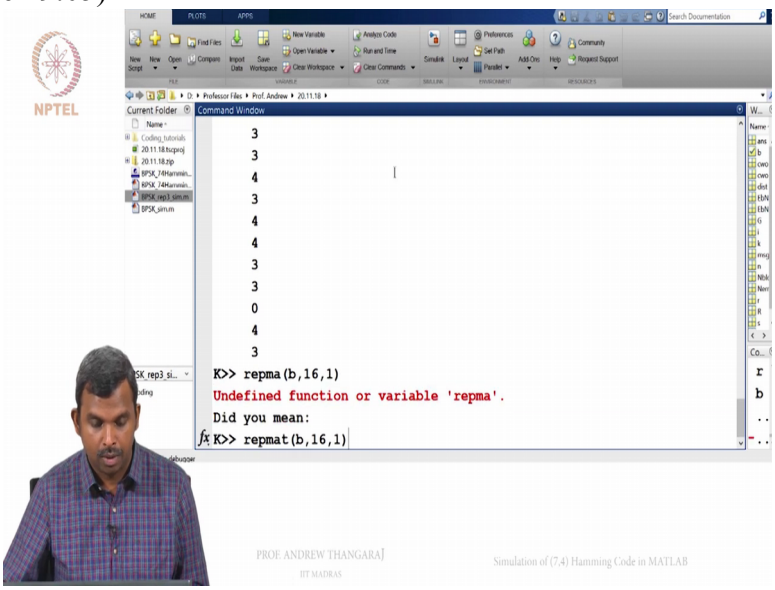
(Refer Slide Time 18:49)




It tells you what the distances are from every possible codeword, Ok.

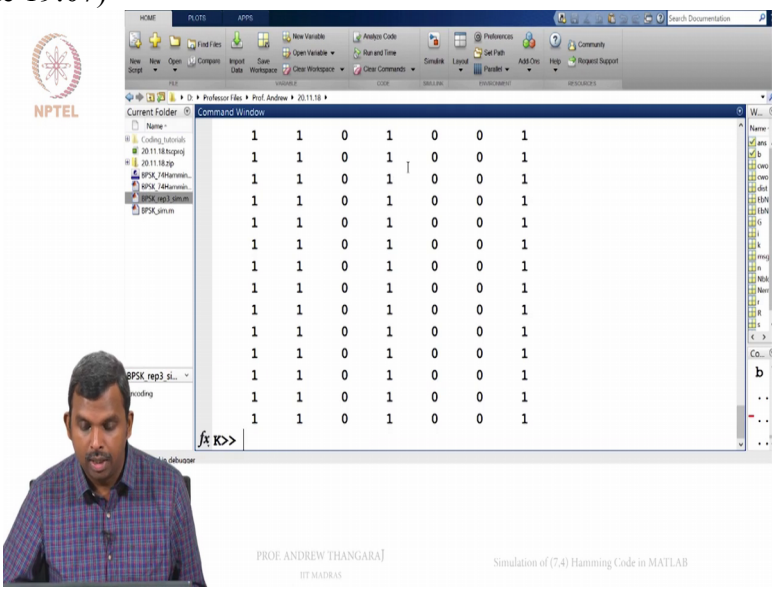
So how did it work? We did this repmat of b comma 16 comma 1,

(Refer Slide Time 19:05)



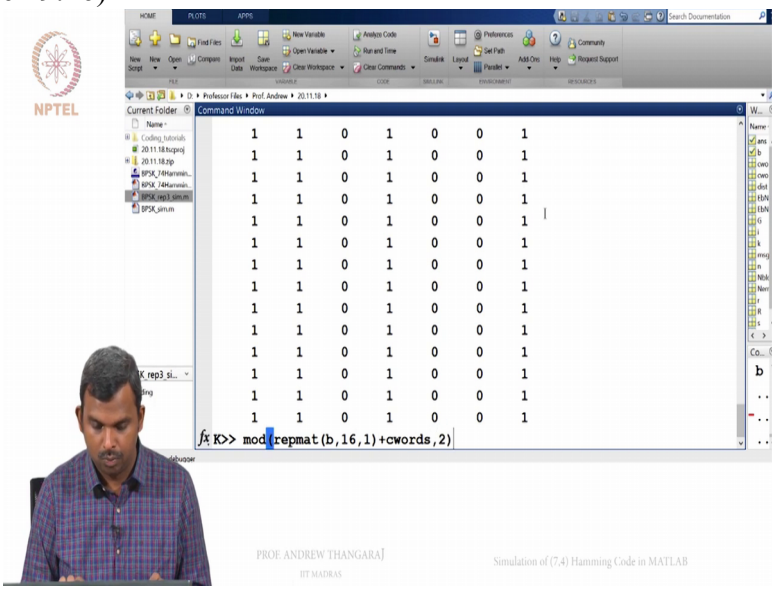

sorry Ok

(Refer Slide Time 19:07)



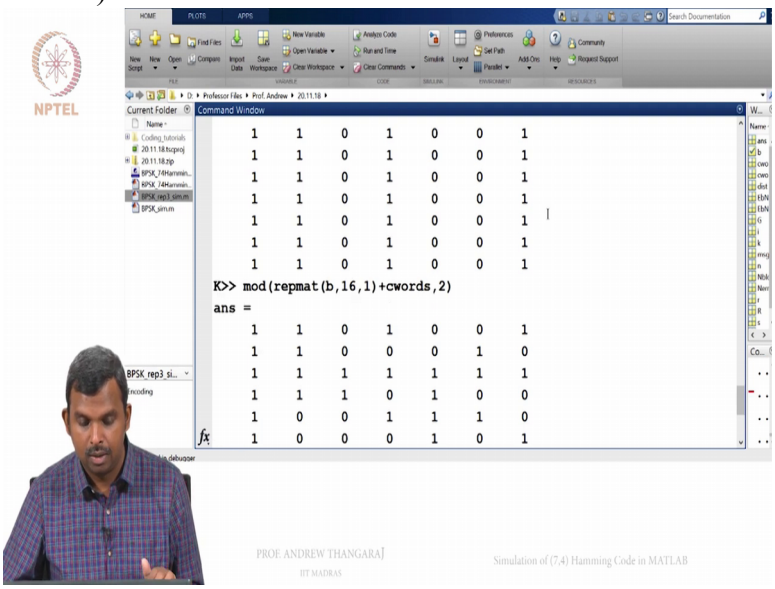

so it repeated the same received vector 16 times. And then I did a XOR of this with my c words,

(Refer Slide Time 19:26)



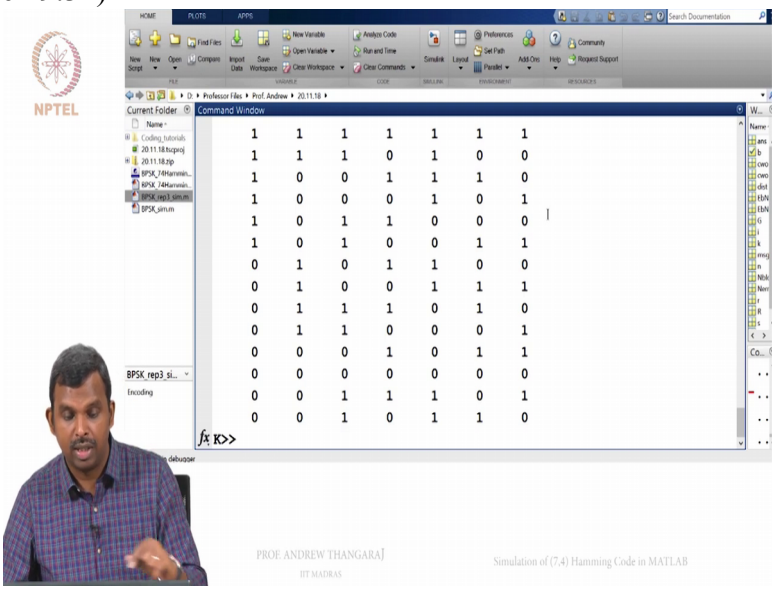

Ok so this gave me all the distances,

(Refer Slide Time 19:29)



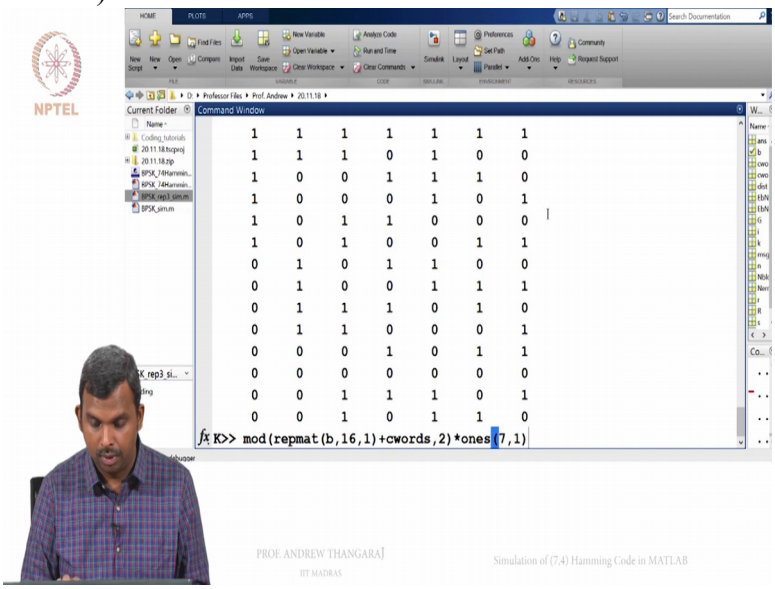

Ok. How do I find the weight of each

(Refer Slide Time 19:31)



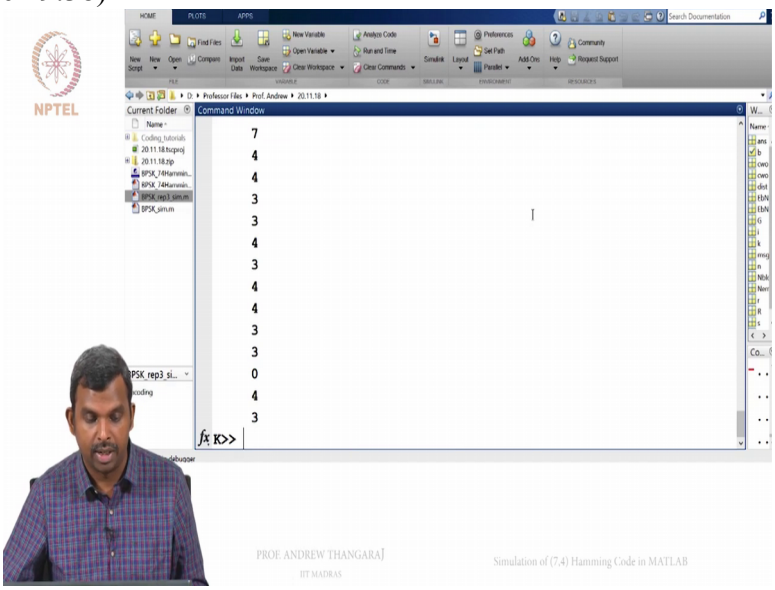

of these guys? I simply multiply the right with 1s of 7 comma 1.

(Refer Slide Time 19:37)




So this gives me this vector,

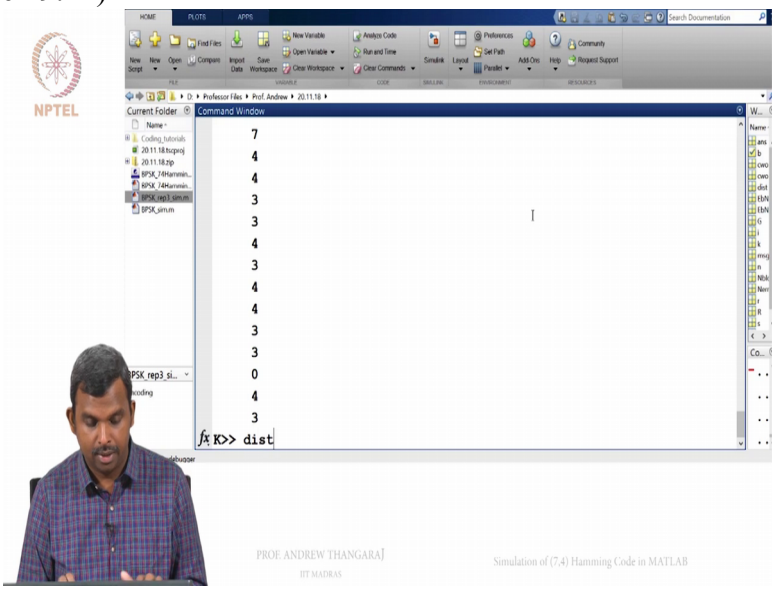
(Refer Slide Time 19:38)




Ok.

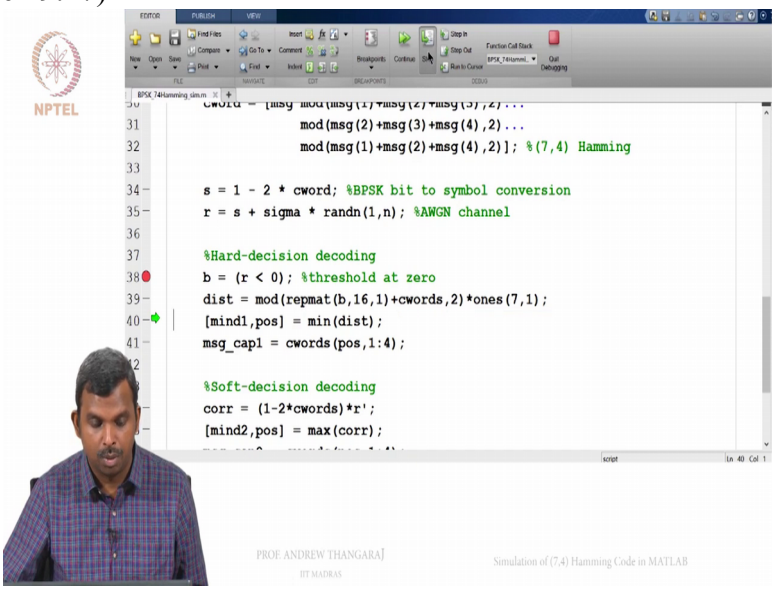
So this is the same as my dist vector,

(Refer Slide Time 19:42)



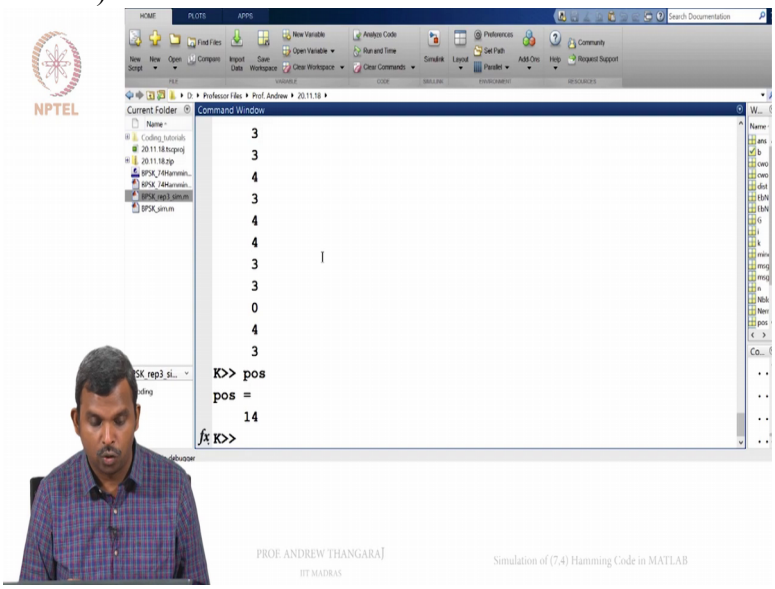

Ok. So the next step finds the minimum

(Refer Slide Time 19:47)



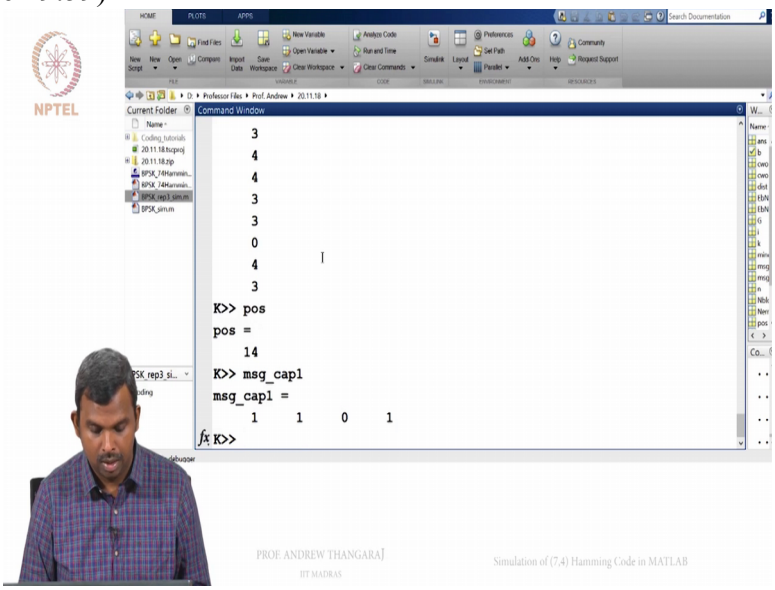

and then sets the message cap. So let us check what it did. The position

(Refer Slide Time 19:55)



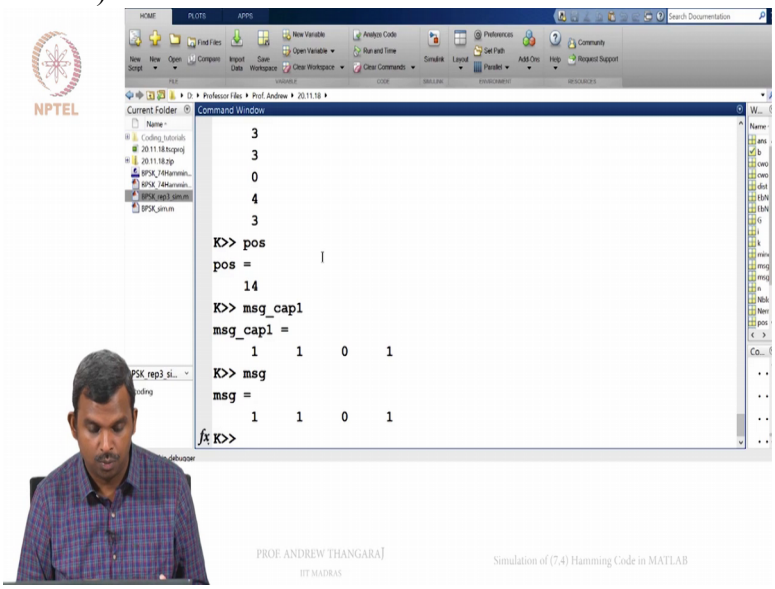

is 14 and message cap 1

(Refer Slide Time 19:59)



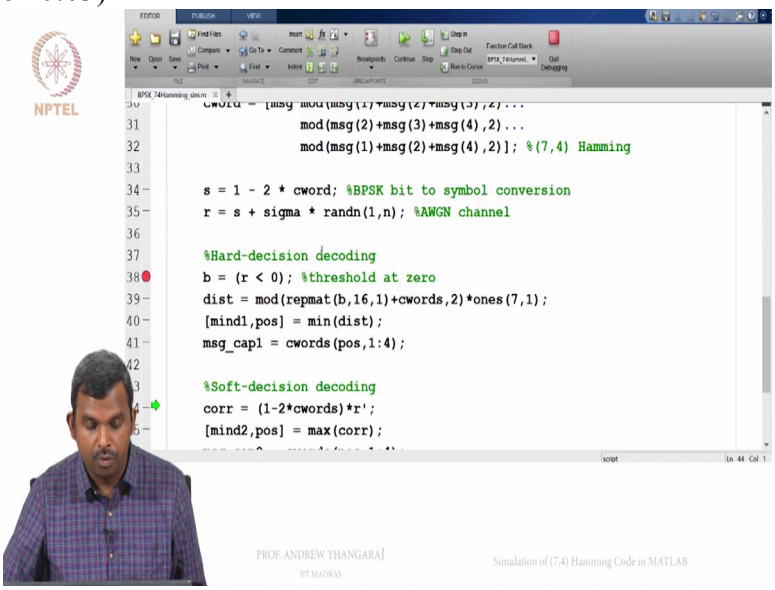

agrees with the message,

(Refer Slide Time 20:00)



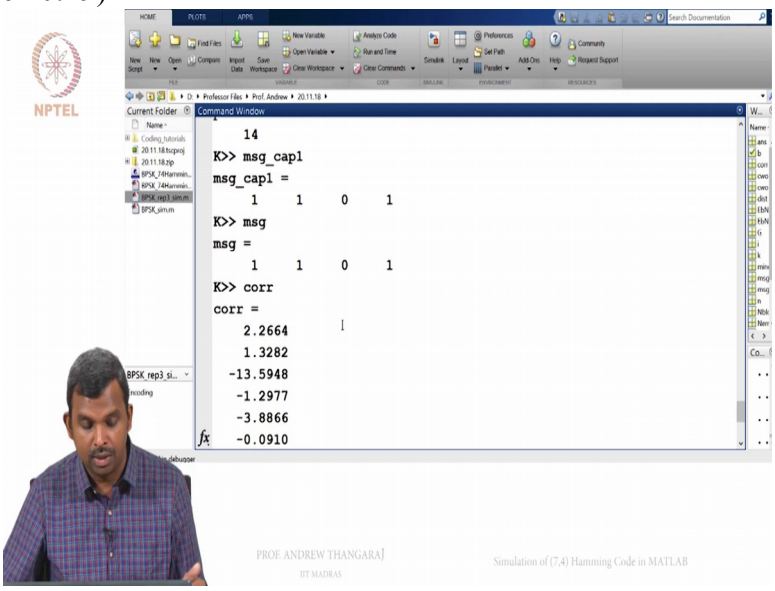

Ok. So this was how the decoder worked.

(Refer Slide Time 20:03)




So let us see if my soft decision decoding is implemented correctly or not. The first step is the correlation, Ok. So remember it works with the soft decision decoding. So correlation will be number, Ok

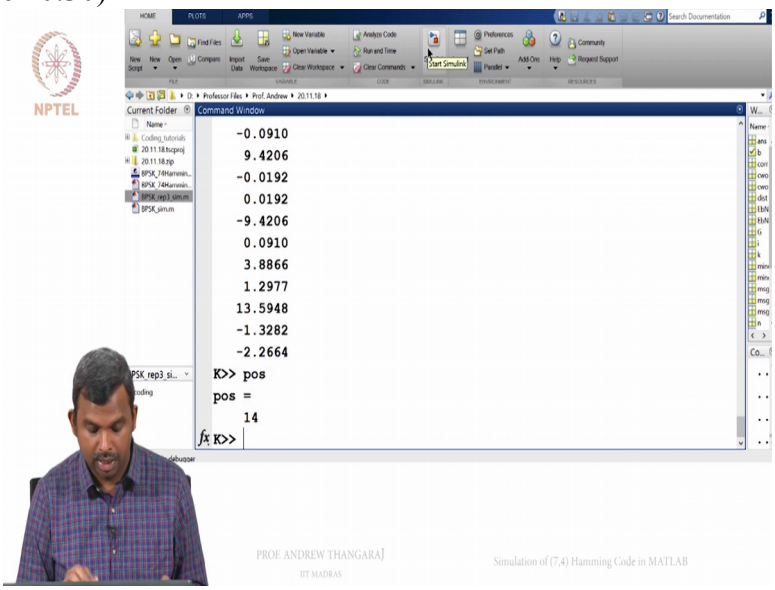
(Refer Slide Time 20:19)




see you get a whole bunch of numbers. The next step is the max of the correlation and assigning message cap.

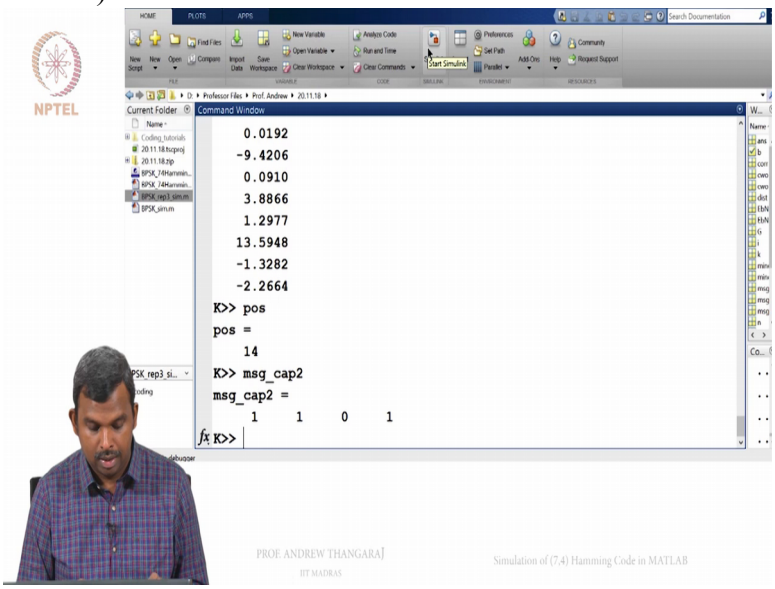
So pos, again

(Refer Slide Time 20:30)



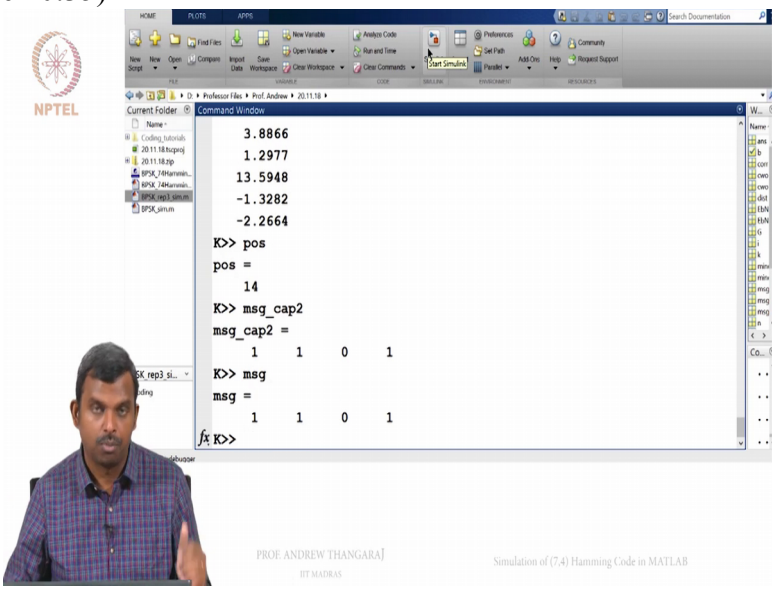

it is 14 and message cap 2

(Refer Slide Time 20:33)



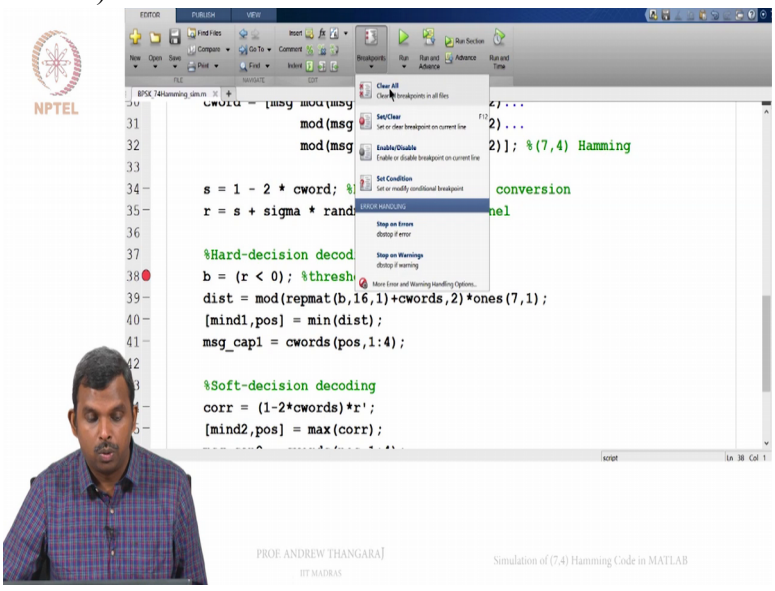

and message also agrees.

(Refer Slide Time 20:35)



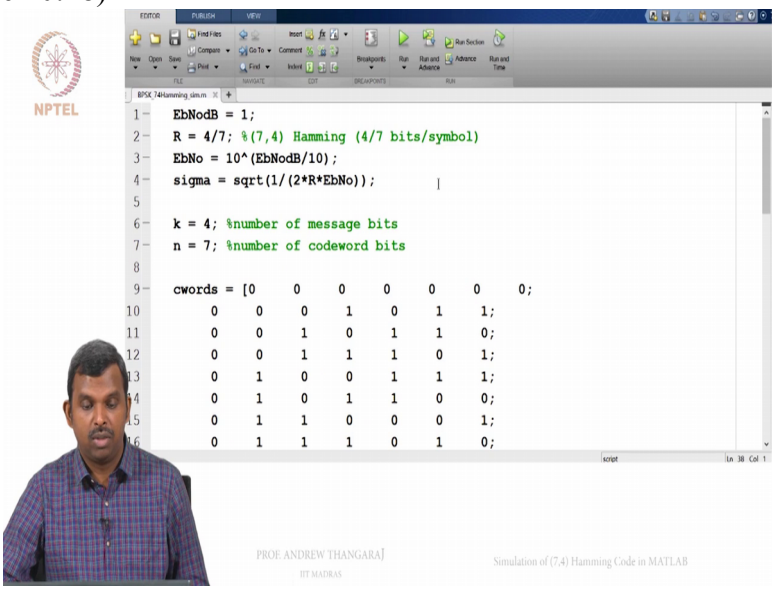

So it is just a easy enough calculation. So this is something that I typically do, just continue and you can quit debugging, Ok.

(Refer Slide Time 20:46)



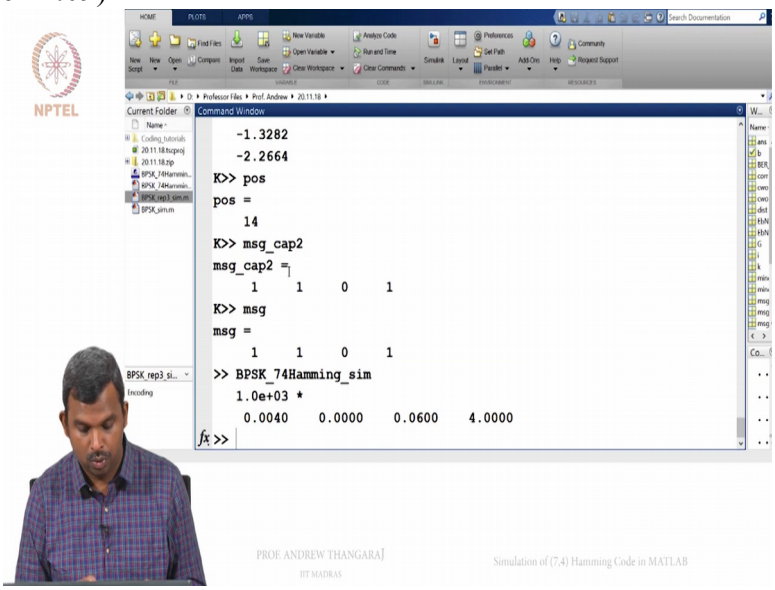

So you can clear this thing, so it looks like

(Refer Slide Time 20:48)



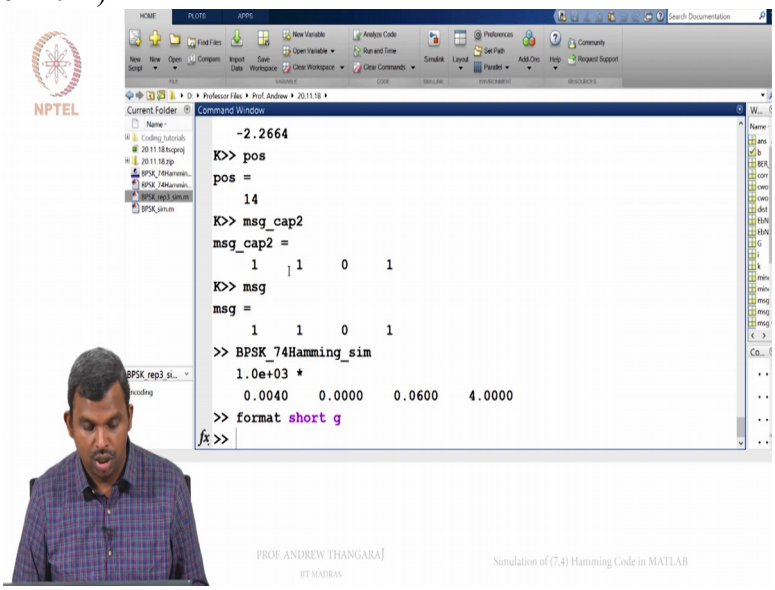

things are working fine. So let us may be pick E b over N naught 4 and then run this then see, use for the hard decision decoder see what happens. So let me just save this. I have E b over N naught 4, I am going to run it now b p s k 7 4 hamming sim

(Refer Slide Time 21:09)



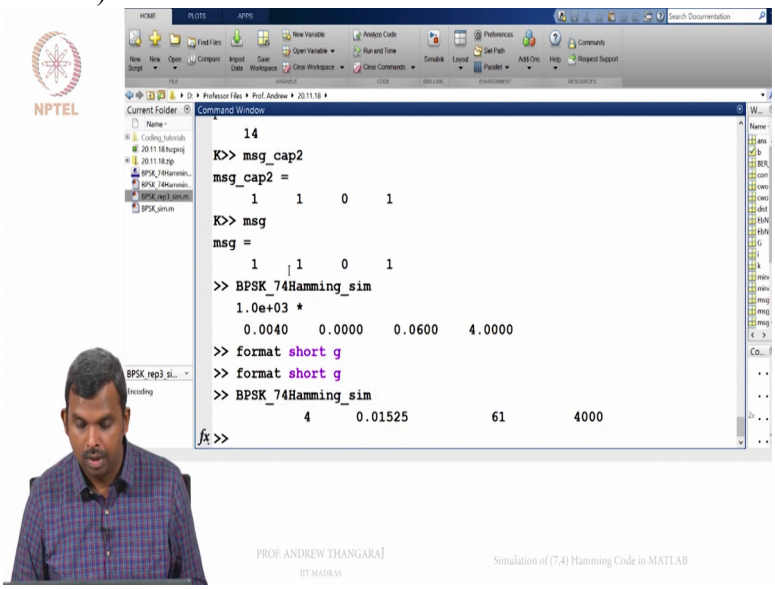

so let us do its format short g,

(Refer Slide Time 21:12)




Ok.

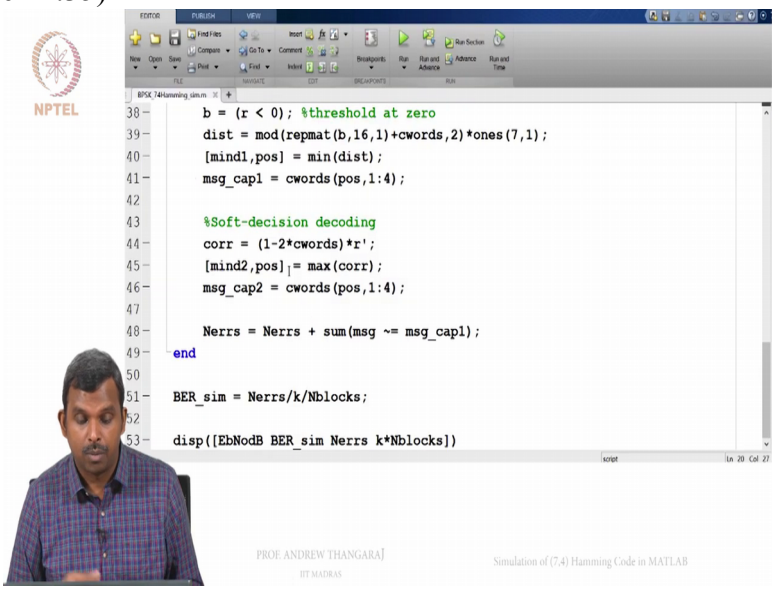
(Refer Slide Time 21:16)




So we get point 0 1, there were 4000 bits that were transmitted. Remember we had 1000 blocks of 4 bits each, codeword is 7000 bits but then, 4000 bits were the message bits. There were 61 bit errors and you got this.

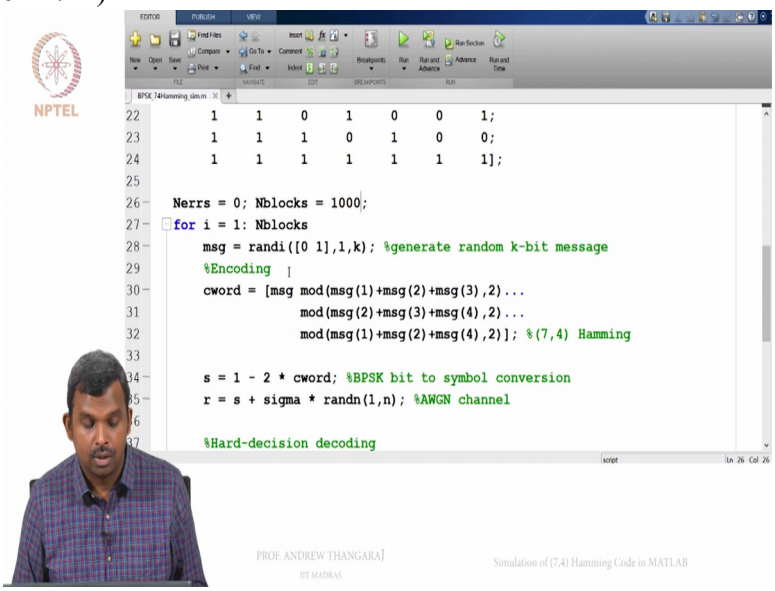
Now it turns out one more error to track

(Refer Slide Time 21:35)




which is very important is the block error bit, Ok. So far we did not have to do it; we were just looking at the message errors.

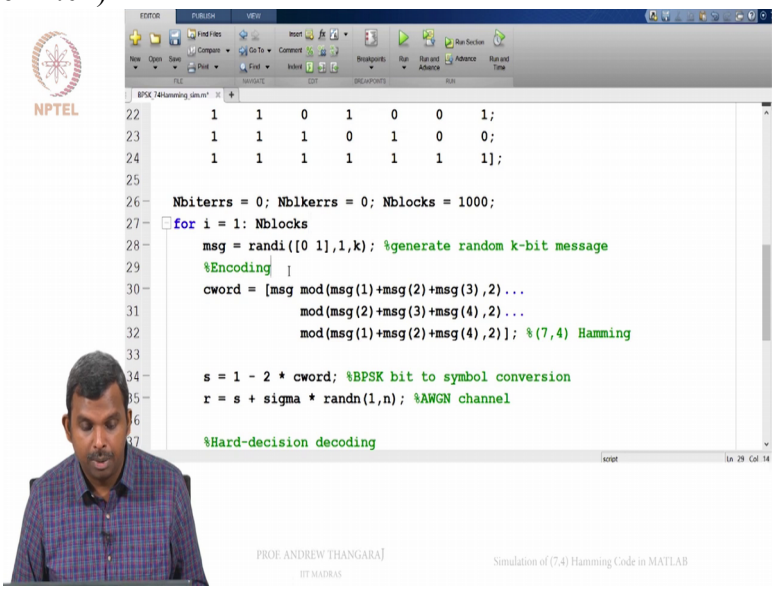
(Refer Slide Time 21:44)




So this I will make a little bit of a modification here. So this I will make as n-bit errors and Matlab does this thing of (()) convert everything, this is good.

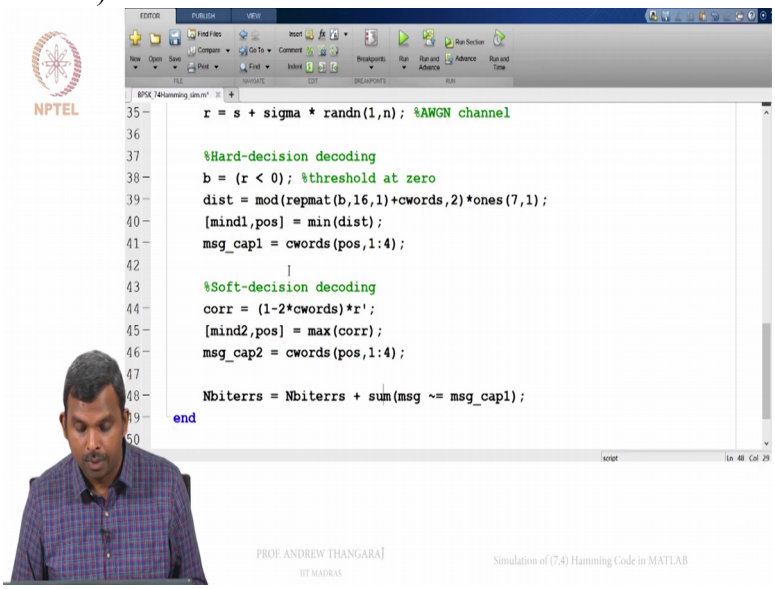
And I will have n-block errors which is also zero, which will be,

(Refer Slide Time 22:01)



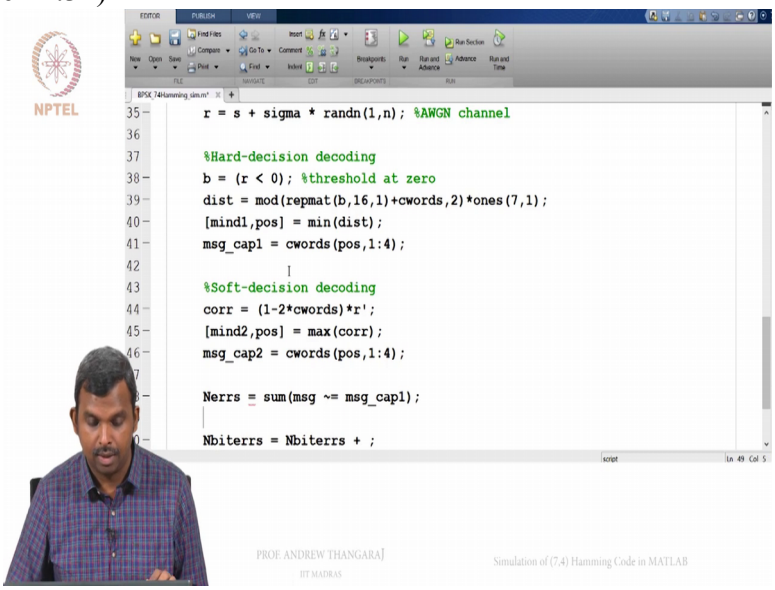

I mean how many codeword errors you have? So what will happen here is,

(Refer Slide Time 22:05)




so I will count this N errors as sum of this guy, Ok. So this is the

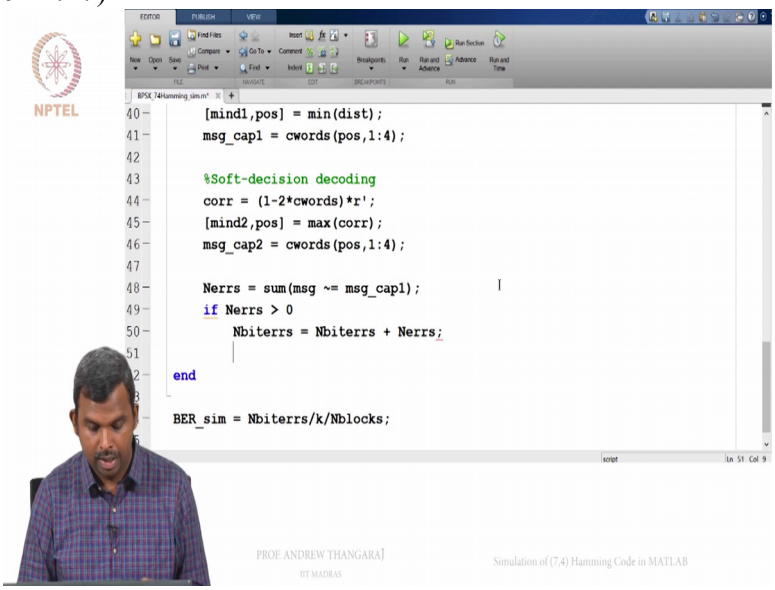
(Refer Slide Time 22:34)




error in one block, Ok.

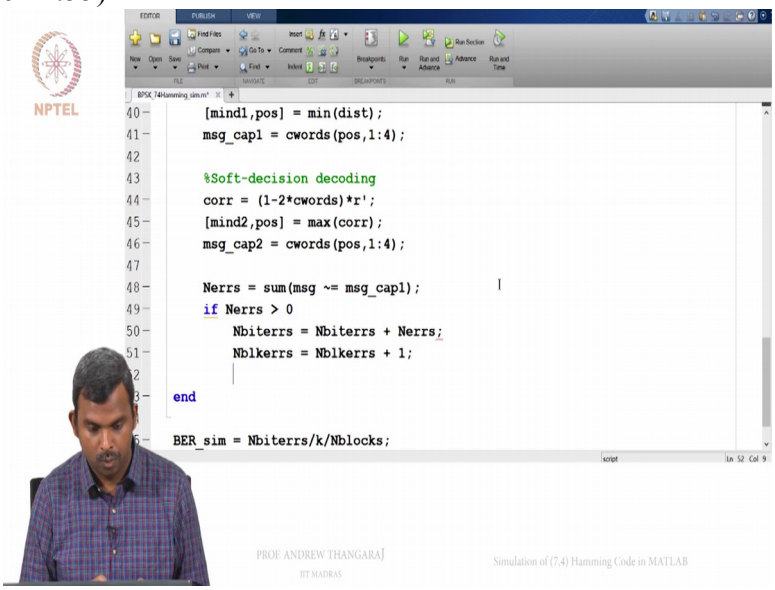
And if N errors is greater than zero, if there is an error I will add N bit errors plus this,

(Refer Slide Time 22:47)



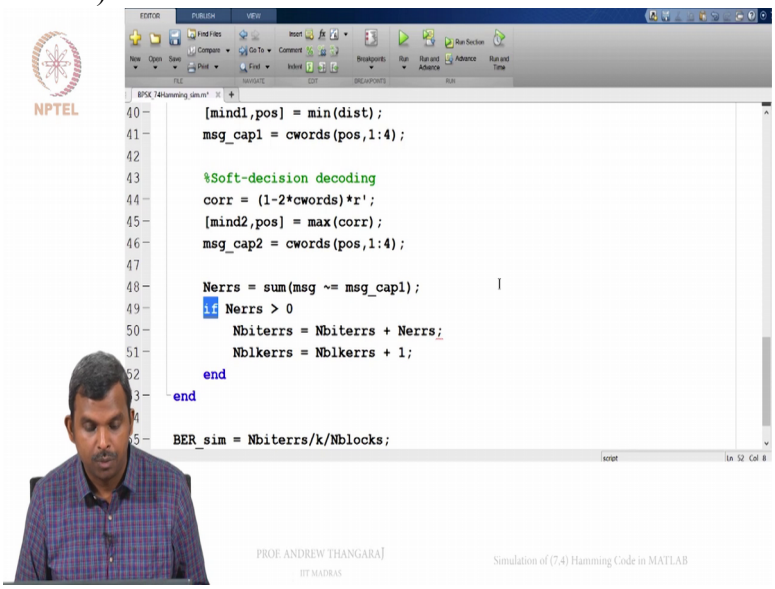

and N block errors equals N block errors plus 1,

(Refer Slide Time 22:55)




Ok.

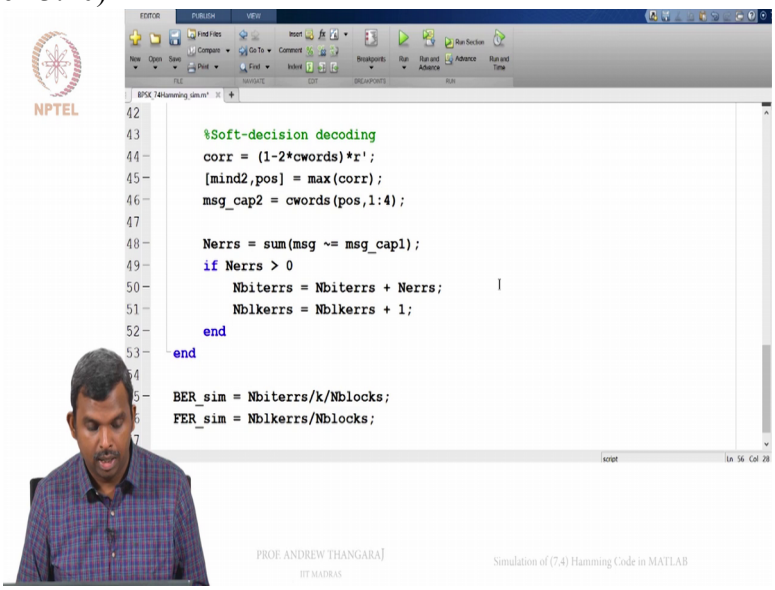
(Refer Slide Time 22:56)




So once again what I am doing here, I am looking at if there was an error in this block. If there was an error in this block I am incrementing N block errors by 1 and then N bit errors by number of errors, Ok.

So this is a simple way. B E R is this, and F E R sim is actually N block errors by N blocks, Ok.

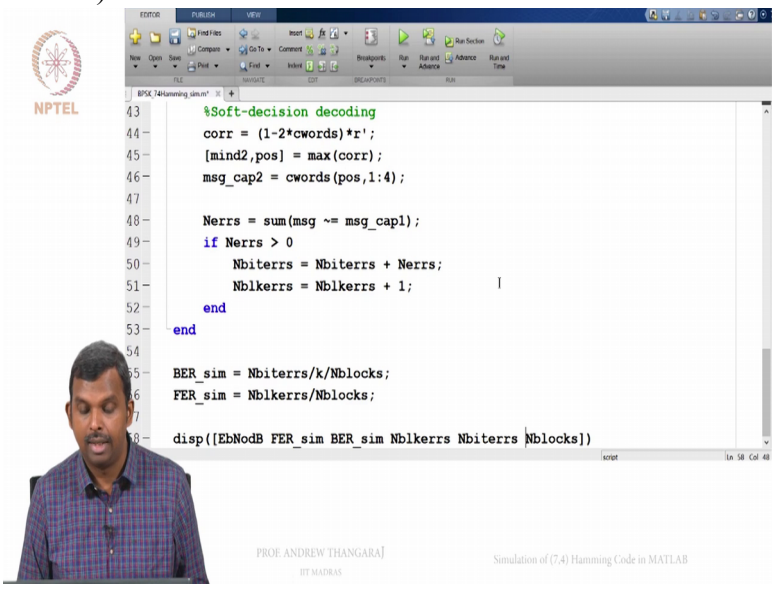
(Refer Slide Time 23:20)




So this is the frame error rate just like I report bit error rates I have to report the, so B E R came out this way, so I will just make a few adjustments here.

N block errors will be output. I will put out F E R sim, B E R sim, bit errors, may be the bit errors can come later, block errors

(Refer Slide Time 23:53)

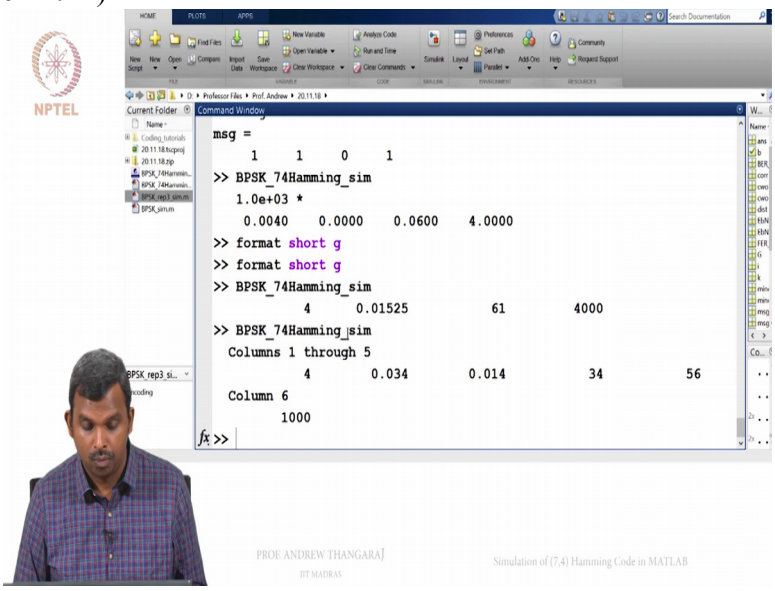



then N blocks. I have made a few changes here. I am tracking block errors as well.

You can see how I track block errors here. I am counting the number of errors in the block. If there were any errors in the block I am incrementing the number of block errors by 1 and number of bit errors by N error.

So this is also something important. B E R, F E R we run so, so let me think if I have made any changes here which are problematic? So let us run it once again.

(Refer Slide Time 24:21)

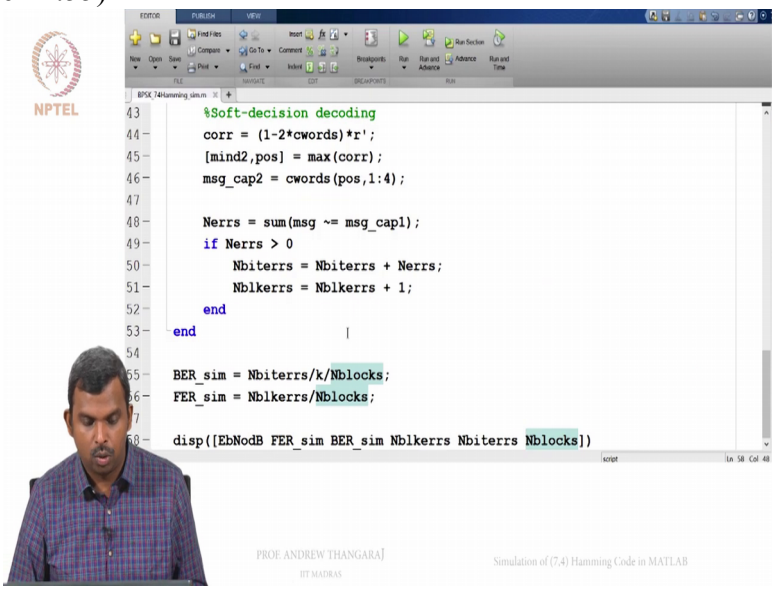



So you get more answers. So there were, so you can see we ran 1000 blocks, Ok. There were 34 block errors, 56 bit errors.

Remember every block error may correspond to one or more bit errors. So number of bit errors will be greater than the number of block errors and the block error rate is point 0 3 4, bit error rate is point 0 1 4 Ok. So as figure of merit you should have at least like 20, 30 blocks in error. Ok, so that is a good number to have to get a good reliable statistic Ok.

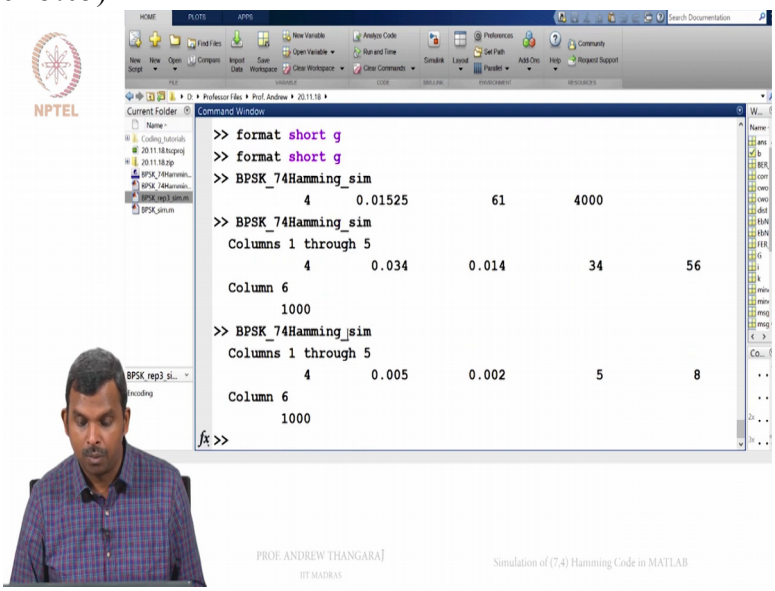
So you can do the soft decision decoder

(Refer Slide Time 24:55)




as well. So how do you I check for the soft decision decoder? I can simply change this to 2, Ok. So if I change this to 2, I should get the soft decision decoder, so let us run the same

(Refer Slide Time 25:05)

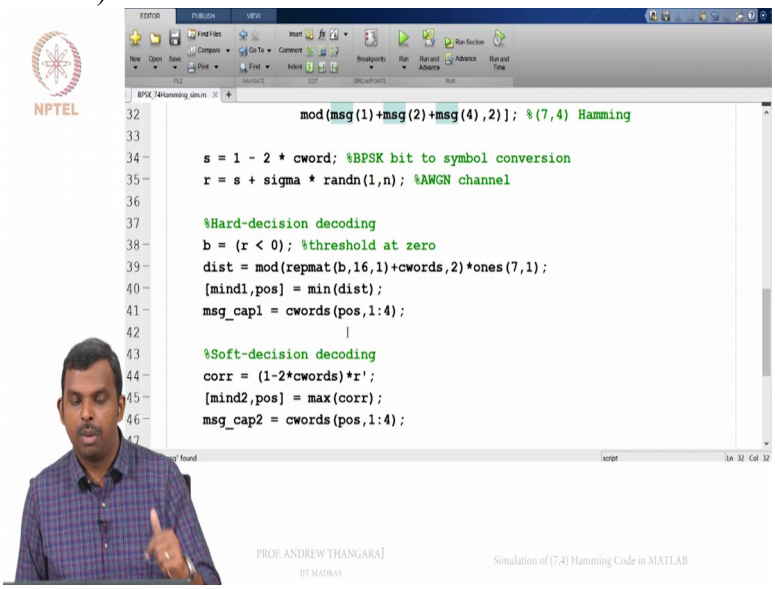



thing and you can see it is significantly better. Ok the soft decision decoder is almost an order of magnitude better than the hard decision decoder.

There were just 5 block errors out of 1000 and bit errors were just very small in number, Ok. So that is, that is the magic of doing soft decision decoding and you can see clearly soft decision decoding is significantly better in the Hamming code, Ok.

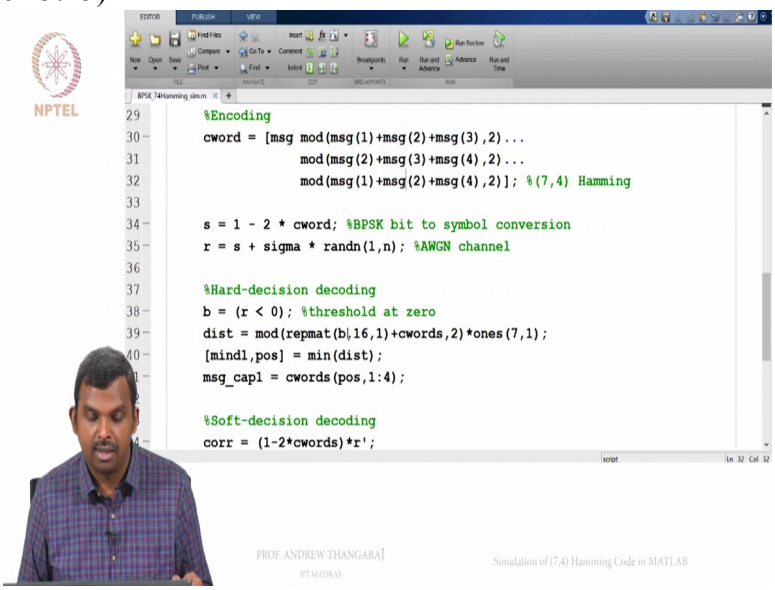
So, so, so you can see how this encoding and decoding has worked for small codes. This is something you can do, for instance the 6 comma 3 example that we saw in the lectures, you can make

(Refer Slide Time 25:41)



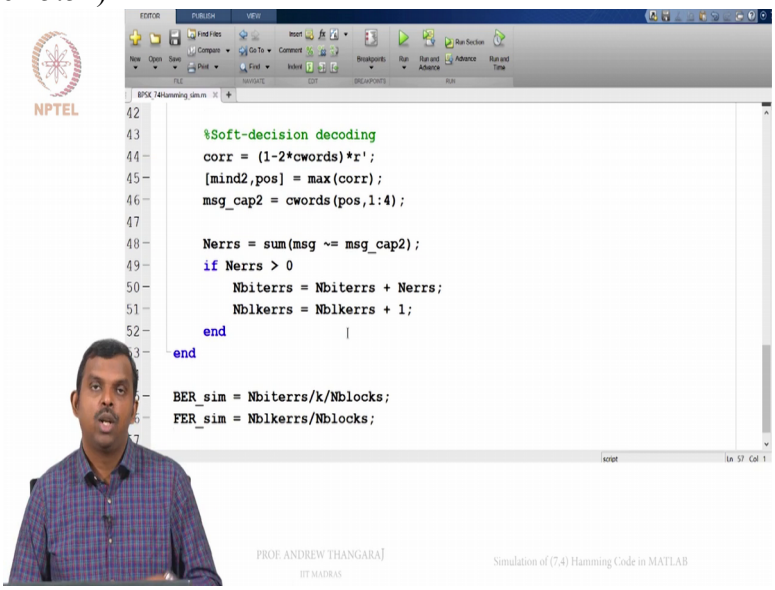

quick modification here to change the encoding and change

(Refer Slide Time 25:45)




the decoding, the list of codewords will have to be changed and you will get your decoder to bug, Ok?

(Refer Slide Time 25:52)




So that is the end of the simulation for 7 4 Hamming code. Hopefully you can take this template and modify it for some other small codes as well. But however if the code size becomes really, really large this is the difficult thing to implement, Ok.

So, so we will stop here for this lecture and the next lecture will be a very nice and important bridge between the modern codes and the starting point of classical codes. We will take 2 codes and look at their decoders in, in very closely and think about them and how they can be used in multiple ways, Ok.

So the two codes we will use are very simple codes, the repetition code and what is called a single parity check code. These two are very crucial. They play a very important role in the bigger decoders, very popular decoders for modern codes. So we will do that in the next lecture. Thank you very much.