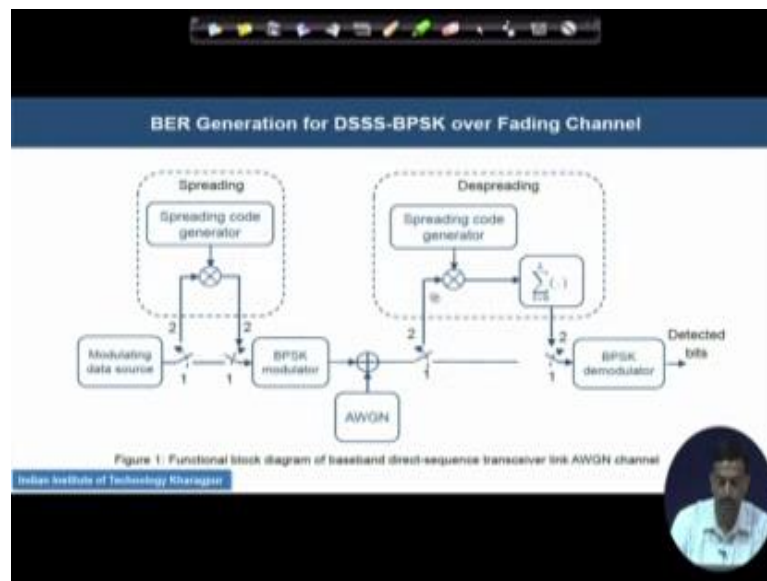


**Spread Spectrum Communications and Jamming**  
**Prof. Kutty Shajahan M**  
**G S Sanyal School of Telecommunications**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 40**  
**Tutorial – V**

Hello friends, so today we shall discuss tutorial five, which basically deals with MATLAB code for BER or bit error rate generation for a BPSK system over AWGN channel, but specifically when the modulated signal is further spread using direct sequence spread spectrum code. So, essentially we will try and evaluate the performance of such a spread spectrum system over a AWGN channel, and try and understand the impact of spreading codes.

(Refer Slide Time: 01:14)



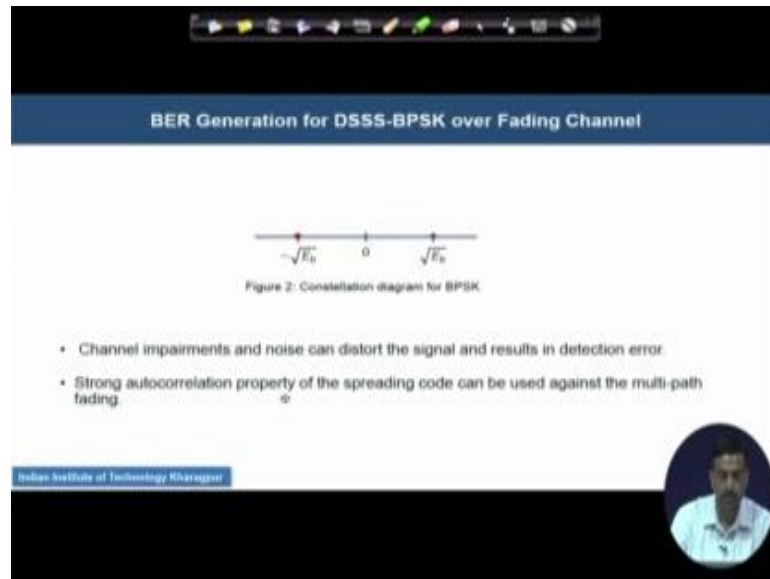
So, what we have here is a block diagram that shows how we are going to proceed with this performance evaluation. So, we first notice that we have modulating data source which will basically be a data stream of binary bits, but in for better understanding, what we do is we represent it in the bipolar form, we have seen this kind of a representation before in our previous tutorial, where we multiply the bipolar data stream with a corresponding bipolar spreading sequence. So, we shall assume bipolar data sequence

over here. And the switch over here is shown in order to indicate that if replace all the switches in position one then what we end up getting is the conventional BPSK in AWGN which was discussed in the previous tutorial. But if all the switches are in position two then what we introduce is the process of spreading as you can see.

So, we will have spreading code generators which will essentially produce the orthogonal spreading code which is multiplied with the bipolar data. And further, it is modulated using the BPSK scheme, the modulated stream then passes through the channel and at the receiving end we have additive white Gaussian noise which is added to this signal causing distortion. Since, we had employed spreading under transmitter we need to have a corresponding scheme for de-spreading the signal. So, we can see the de-spreading block here.

The received signal corrupted with noise is applied through a multiplier and the same spreading code which we had used at the transmitter is once again used at the receiver. Once the multiplication is done, we then sum up the contributions of each block of spread code and we then make a decision as to what is the now demodulated bit. So, we use a BPSK demodulator. I think this straight chain was explained in the last tutorial, so that is pretty straight forward, but the introduction now is of the spreading code generator over here that is transmitter as well as the receiver which is used for spreading and de-spreading respectively. So, let us see how we can implement this kind of a system, how we can plug in this to modules, and what kind of performance it yields.

(Refer Slide Time: 04:32)



The slide features a blue header with the text "BER Generation for DSSS-BPSK over Fading Channel". Below the header is a white area containing a constellation diagram for BPSK. The diagram shows a horizontal axis with three points: a red dot at  $-\sqrt{E_b}$ , a black dot at  $0$ , and a red dot at  $\sqrt{E_b}$ . Below the diagram is the caption "Figure 2: Constellation diagram for BPSK". Underneath the caption are two bullet points:

- Channel impairments and noise can distort the signal and results in detection error.
- Strong autocorrelation property of the spreading code can be used against the multi-path fading.

At the bottom left of the slide, there is a blue box with the text "Indian Institute of Technology Kharagpur". At the bottom right, there is a circular inset image of a man in a white shirt.

Just to recall and this was I think also discussed in the previous tutorial that BPSK constellation diagram basically looks like this, where we have two possible symbols depending on no phase shift and a phase shift of 180 degrees. So, the idea is to use the strong auto correlation property of a spreading code, and see as to what impact has on multi path fading. However, please note that in this particular tutorial we are not going to discuss multi path fading, we will merely see the impact of spreading on an AWGN channel with no fading for say. So, this is something that we will deal within the next tutorial, but this is just mentioned over here, because it is an important point to be noted.

(Refer Slide Time: 05:25)

```
1 % This code is for the BER of the system.
2 % The code is for the system.
3 % The code is for the system.
4 % The code is for the system.
5 % The code is for the system.
6 % The code is for the system.
7 % The code is for the system.
8 % The code is for the system.
9 % The code is for the system.
10 % The code is for the system.
11 % The code is for the system.
12 % The code is for the system.
13 % The code is for the system.
14 % The code is for the system.
15 % The code is for the system.
16 % The code is for the system.
17 % The code is for the system.
18 % The code is for the system.
19 % The code is for the system.
20 % The code is for the system.
21 % The code is for the system.
22 % The code is for the system.
23 % The code is for the system.
24 % The code is for the system.
25 % The code is for the system.
26 % The code is for the system.
27 % The code is for the system.
28 % The code is for the system.
29 % The code is for the system.
30 % The code is for the system.
31 % The code is for the system.
32 % The code is for the system.
33 % The code is for the system.
34 % The code is for the system.
35 % The code is for the system.
36 % The code is for the system.
37 % The code is for the system.
38 % The code is for the system.
39 % The code is for the system.
40 % The code is for the system.
41 % The code is for the system.
42 % The code is for the system.
43 % The code is for the system.
44 % The code is for the system.
45 % The code is for the system.
46 % The code is for the system.
47 % The code is for the system.
48 % The code is for the system.
49 % The code is for the system.
50 % The code is for the system.
51 % The code is for the system.
52 % The code is for the system.
53 % The code is for the system.
54 % The code is for the system.
55 % The code is for the system.
56 % The code is for the system.
57 % The code is for the system.
58 % The code is for the system.
59 % The code is for the system.
60 % The code is for the system.
61 % The code is for the system.
62 % The code is for the system.
63 % The code is for the system.
64 % The code is for the system.
65 % The code is for the system.
66 % The code is for the system.
67 % The code is for the system.
68 % The code is for the system.
69 % The code is for the system.
70 % The code is for the system.
71 % The code is for the system.
72 % The code is for the system.
73 % The code is for the system.
74 % The code is for the system.
75 % The code is for the system.
76 % The code is for the system.
77 % The code is for the system.
78 % The code is for the system.
79 % The code is for the system.
80 % The code is for the system.
81 % The code is for the system.
82 % The code is for the system.
83 % The code is for the system.
84 % The code is for the system.
85 % The code is for the system.
86 % The code is for the system.
87 % The code is for the system.
88 % The code is for the system.
89 % The code is for the system.
90 % The code is for the system.
91 % The code is for the system.
92 % The code is for the system.
93 % The code is for the system.
94 % The code is for the system.
95 % The code is for the system.
96 % The code is for the system.
97 % The code is for the system.
98 % The code is for the system.
99 % The code is for the system.
100 % The code is for the system.
```

So, let us look at the MATLAB code, which facilitates implementation of the system that we just saw. So, initially we can see here that we have the number of samples generated. So, something like 10, 3 into 10 to the power of 5 samples it should be sufficiently large of course. We have a length of the spreading code that can be specified. So, this is a very general code that can be used for any length. So, in this case we have chosen a length of four we set the range of  $E_b/N_0$  values from minus 2 to 12 dB and in increments of 2 dB each. And then we have case switch wherein basically we can select whether we want to plot the final BER in terms of  $E_b/N_0$  on the x-axis over SNR.

So, we shall see what is the relevance of this particular statement that is we shall see the results plotted both the axis  $E_b/N_0$  as well as SNR, and see whether there is any difference between the two line number fourteen over here is the generation of the spreading code. So, there is a inbuilt function library function in MATLAB which can generate Hadamard codes provided you specify the length of the code that you desire to generate. So, we have a code length of 4. So, this will basically generate matrix basically, where in we can it will basically a 4 by 4 matrix where we have three of those rows which will which can be used as Walsh Hadamard codes which are of course, orthogonal to each other. The last row of course, is rendered because it will be a row of all ones. So, please try this out separately this commands separately in MATLAB in order to see what

this yields.

You can just ignore these two lines for the time being, because this was written for some other purpose. So, what we see now is that the data stream that we wish to generate this is given by the number of the samples if we require so many bits basically. So, this line number 17 as you may be familiar with was discussed in the last tutorial is to generate the bipolar sequence or random sequence that is. So, it will generate a stream of plus 1 or minus 1. We spread the data by using the chronicle function, which is nothing but what we do is we select any one row as we see from this Hadamard matrix which was generated in line number 14. So, we select as I said the first three rows are useful for to be used as orthogonal code basically. So, we choose this third row, you could choose the first row as well as the second row, and try out the same exercise.

And we basically multiply each of those that is the each row that is third row with the incoming data which is a bipolar data. So, essentially line number 18 nothing is multiplying each bipolar entity generated in 17 with a code, which is selected as a particular row in this Hadamard matrix. So, once this is done, we basically declare this whatever we got over here as transmit data. And now this is nothing but the spread baseband transmit signal. We calculate the power of this data because this will be later on be used in my number 13 in order to normalize noise, so that is the straight forward calculation we just take the data stream find out its absolute value square. And we divide it by the length of the data in order to normalize the power.

We also calculate rather we store the length of this transmit data that is a spread data as a variable N because this is also going to be used perhaps later on. Now, we run this program for rather we loop it over this entire range of E b by N naught. So, therefore, whatever length of this vector which we generated over here we run it over that length, so as to calculate the bit error rate for each value of E b by N naught. We convert this E b by N naught to a linear value or rather to the linear scale in line number 26, and we actually now we will see the plot of rather how the program works.

So, depending on whether we select E b by N naught or whether we select the SNR, we

accordingly add noise. And as you can see in noise is basically added as in terms of the square root of power, this is in order to normalize noise with respect to the data. And it also has to be the length of the code needs to be taken into account while calculating the noise. And of course, we divide it by a root of twice  $E_b$  by  $N$  naught linear because the idea is to control the increment of this  $E_b$  by  $N$  naught by varying noise rather than varying the signals, so that is what the control is we generate the noise that is additive white Gaussian noise using this expression. It has variance of 1 and mean of 0.

So, essentially these two the lines of code are the same with the small difference that this is done all with respect to the  $E_b$  by  $N$  naught, whereas this is to be done with respect to the SNR. So, once this noise is generated, we simply add the noise to the transmitted data that is the spread data this is nothing but the received signal. So, this is where the additive operation happens. Now, the receiver part of the signal what we do is we take the received data which is nothing but transmitted data corrupted with noise and we reshape this data.

Reshaping is nothing but arranging because this is going to be vector. So, now, what we do is we arrange it in the form of some matrix with number of rows is equal to the length of the code, and number of columns equal to the length of the received data dividing by the length of the code. So, this is nothing but to convert reshape is nothing but a command to convert a vector into a matrix with these many number of rows that is let us say if we denote this by some  $m$  and this is denoted by some let say  $n$ . So, what we will what we will get basically is  $m$  comma  $n$  matrix.

And then we take the transpose of that matrix, so that we get a  $n$  comma  $m$  matrix and this matrix will have nothing but its rows of that is the rows of this matrix will correspond to each data bit which is spread. So, it will correspond to a set of spread bits which corresponds to  $m$  over which is equivalent to a every data bit that you have transmitted. So, once we get it in that form, what we do is we find a size of this matrix. So, it is nothing but suppose I had let say 4 data bits and if I had the length of the code length of course, is already specified and that is equal to 4. So, what I will have is totally 16, 4 into 4 - 16 will be the length of this received data and then I will break it up into 4

by 4 matrix.

And so what I will have is the length of sorry the size even if I take the transpose it still be 4 by 4 and the size of this matrix will the number of rows will be equal to 4. So, we make use of that over here in generating basically a column vector of ones, and we choose the same code which we had used at the transmitter that is in line number 18 to spread a signal. And this chronicle operation is basically to repeat the same code, and since it repetition because it is being operated on a chronicle basis with ones. So, what this temp one signal will look like is, it will look like the same code in column form repeated number of time that is here in this case  $x$  is equal to 4. So, it will be repeated 4 number of times. So, this line number 47 essentially generates a matrix and the matrix will comprise of columns which are repeated of the same code means if the same code repeated.

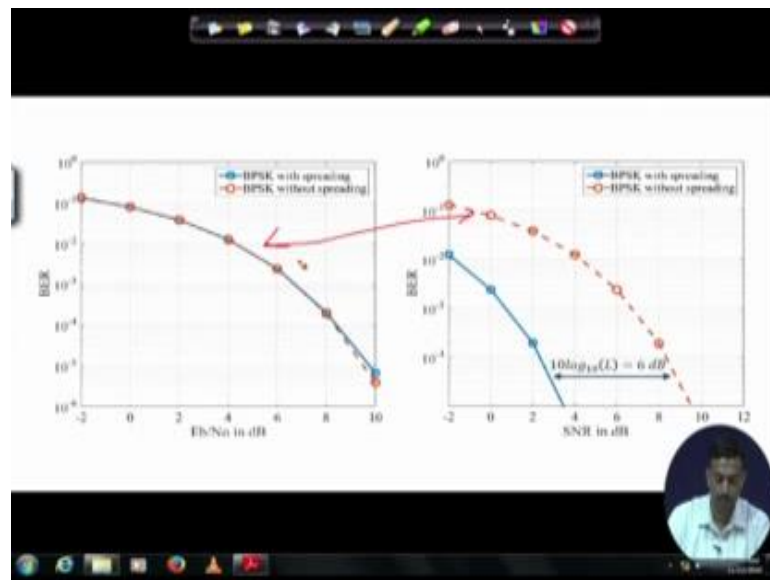
So, what we then do is we multiply the temp signal, which we had originally obtained where we are segregated the received signal vector into the corresponding spread vectors. So, we multiply the two that is we multiply the signal with the code. So, every set of spread bit signals are multiplied with the same code, which is essentially just to go back, so which is nothing but the de spreading operation.

If you remember the signal that is received at the receiver should be multiplied with the spreading code once again, so that is precisely what we do in this line over here that is line number 48. So, finally, the summation operation is required, because every set of de-spread signal corresponding to a given bipolar entity should be summed up that too was highlighted in the block diagram before that is the summation operation over here, so that corresponds to line 49 nine. And once we get the de-spread signal all that is left is to demodulate. So, I think there is a small error over here. So, this should not be QPSK, I am sorry this should be BPSK. So, just pardon me for this error.

So, what we do is we de-spread the signal that is sorry demodulate the signal de spreading is in 49. So, in 50, we demodulate the signal. And we basically find out how many demodulated bits differ from the original bits that we have transmitted that is in

line number 17. So, the difference is nothing but if there is a difference, then it will record a one and we sum up the number of such ones in order to find out how many bits is in error. And finally, the probability of error is nothing but the number of bits divided by sorry the number of bits in error divided by the number of total number of bits which was transmitted. So, then we plot the error that is the probability of error versus both  $E_b/N_0$  as well as SNR, but in this particular program we have done it versus  $E_b/N_0$  and let us see what is the impact of doing that.

(Refer Slide Time: 19:02)



So, what we see is that looking only at  $E_b/N_0$  in dB, we see that the bit error rate curve for BPSK with or without spreading. So, without spreading was dealt with in the last tutorial. And the result for BPSK we are with spreading that is with the DSS spreading is shown over here in blue. So, we see that it is essentially the same that is because even if we spread the signal the energy in 1 bit now is spread over 4 bits. So, essentially there is no difference as far as BER performance is concerned due to conservation of energy that is the main point.

So however, if we plot the same performance versus SNR in dB, since here energy was spread over from one bit to four bits in this particular case, but power basically remains the same because in power when we when we talk about signal power there is no notion



of time. So, spreading if we try to simulate it the power across the 4 bits will remain the same which depends upon the amplitude. So, here we actually see the difference between the performance of the BPSK with and without spreading. So, this is; what is the conventional curve that we get if you notice these two curves are same that is  $E_b$  by  $N_0$  and SNR without spreading is the same. However, with spreading you will see a gain as far as plot in SNR is concerned and that gain is equal to the processing gain in dB. So, in this case  $L$  was equal to 4, so  $\log$  to the base 10  $L$  turns out I think 0.6, so that into 10 is about 6 db.

So, we clearly see that there is a dB gain when we plot this in SNR terms. However, the true picture is obtained when we talk in terms of  $E_b$  by  $N_0$  and we see that the performance is the same with or without spreading in fact, the performance impact can be more appropriately judged in the presence of interference, because spreading because this particular performance evaluation is only for noise. And for say there is no apparent impact of BPSK with spreading in AWGN noise; however, we know that spreading does provide interference margin and gain. So, the presence of interference there will be some gain expected we shall deal with in some other tutorial when we discuss multi user in CDMA systems.

So, in a summary, what we have seen today is that the signal when spread with a spreading code and in this case Walsh Hadamard code and then de-spread using the same code yields us a  $E_b$  by  $N_0$  versus BER performance with this the same without spreading. But of course, the important and the crucial thing is that we have made an assumption that the transmitter and receiver are perfectly synchronized and that assumption has lead us to appropriately decoding the original signal. You could also try and introduce a delay and see what happens on the performance will degrade substantially if the synchronization assumption is violated.

So, this completes tutorial five, I would advise you to go through these codes carefully and try or change certain parameters including the length of the code try and generate higher length codes and see the effect of same on the BER. And possibly also try different types of codes; in this case, we had used Hadamard code, so you could use some

other code. And I hope you have understood and appreciated the difference between the plots of BER in terms of  $E_b/N_0$  and in terms of SNR. Just to reiterate energy conservation principle is highlighted over here. As I said the energy over a single bit basically gets spread over 4 bits and that is why there is no change in BER performance, but performance difference becomes obvious when we try and plot it against SNR, but the true measure is in terms of  $E_b/N_0$ .

Thank you.