Digital Communication using GNU Radio

Prof. Kumar Appaiah

Department of Electrical Engineering

Indian Institute of Technology Bombay

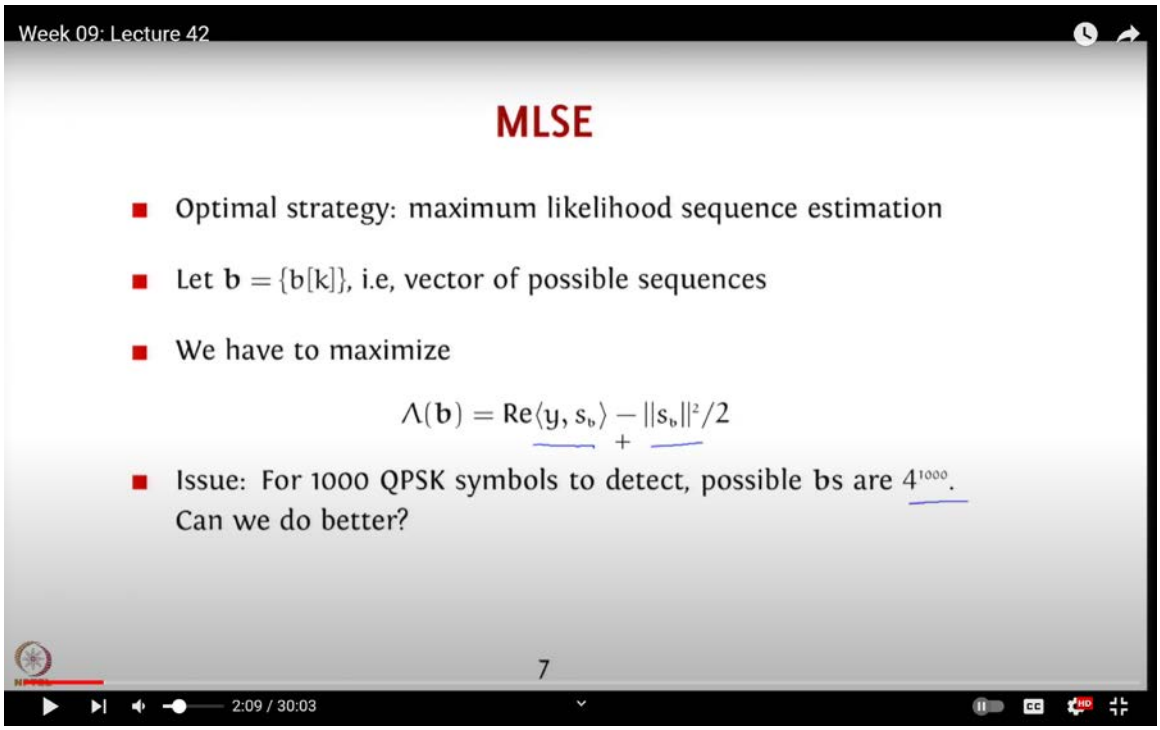
Week-09

Lecture-42

Detection Strategy for Dispersive Channels

Hello, and welcome to this lecture on digital communication using GNU Radio. My name is Kumar Appiah, and I am from the Department of Electrical Engineering at IIT Bombay. Today's lecture continues our exploration of maximum likelihood sequence estimation (MLSE). In our previous lecture, we discussed the challenge of detecting a sequence of symbols transmitted through a channel with a convolutional characteristic. Essentially, due to the mixing of symbols in the channel, we experience what is known as intersymbol interference (ISI).

(Refer Slide Time: 02:09)



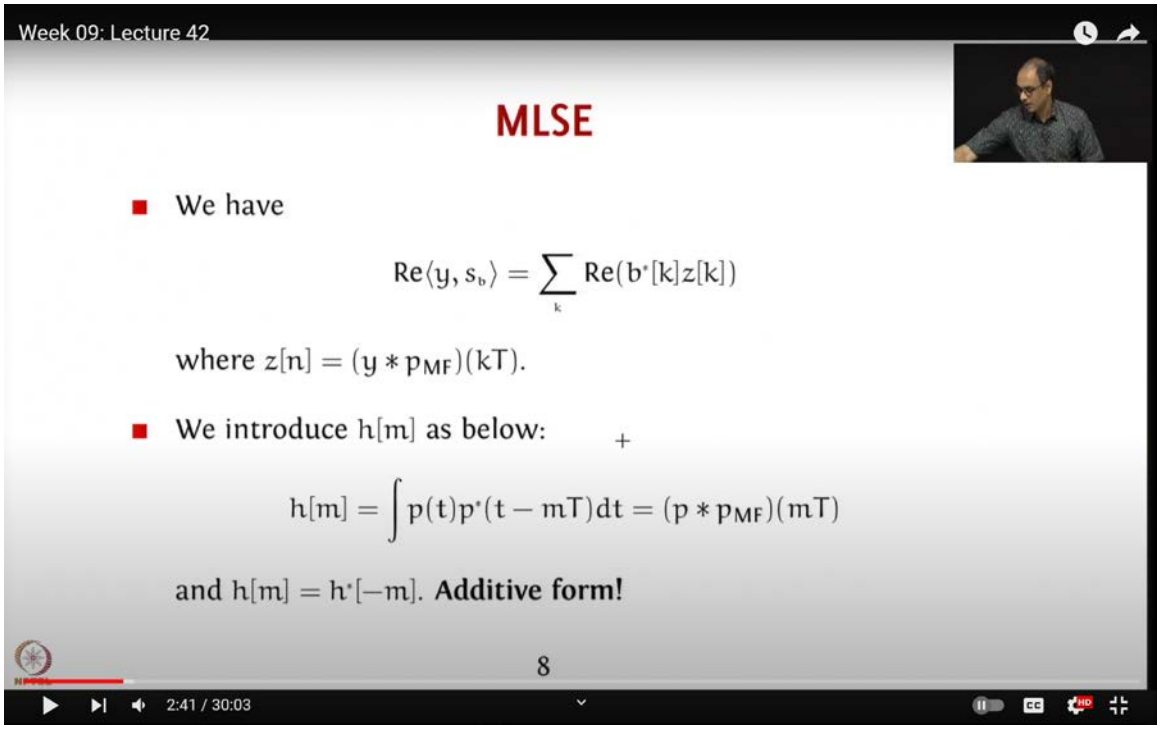
To recap, we developed a noise model and formulated an optimal detection strategy, which ultimately leads to optimizing a particular expression. This expression, denoted by Λ_B, is given by:

$$\Lambda_B = \mathrm{Re}(\langle y, S_B \rangle) - \frac{|S_B|^2}{2}$$

Here, S_b represents the received signal, which embodies the combined effects of the channel, including its convolutional nature, and y is the observation. The real challenge lies in calculating all possible combinations of S_b, as they account for both the channel and modulation.

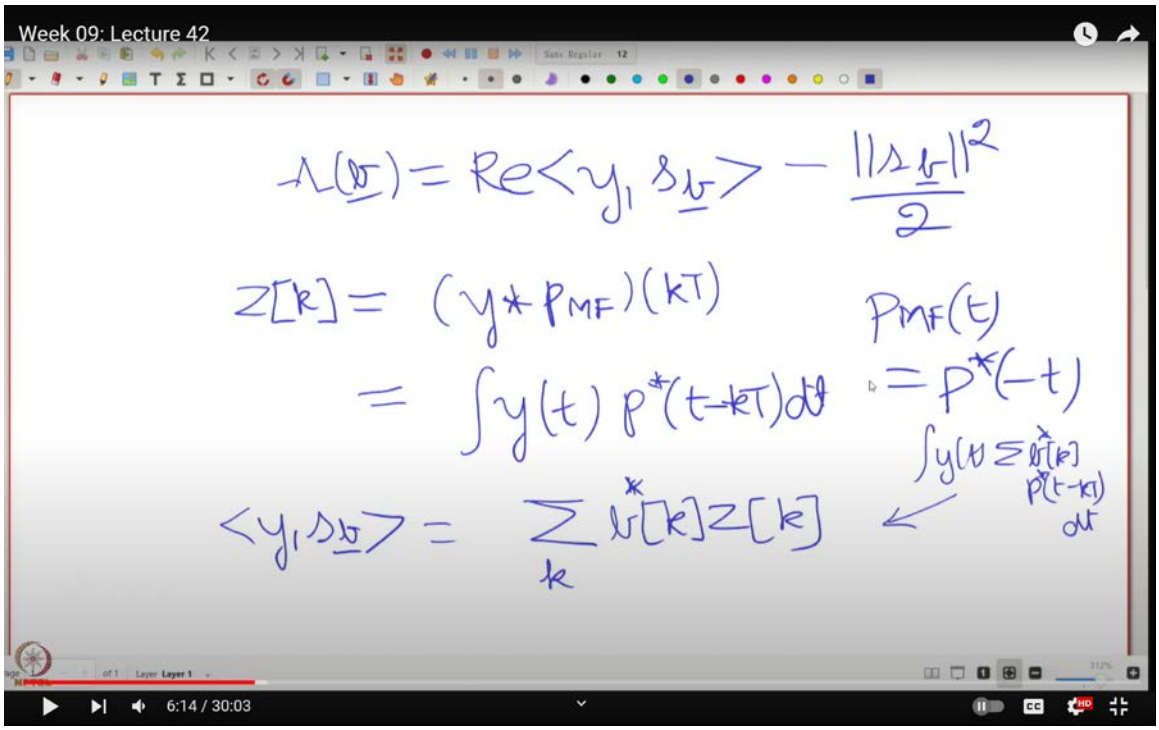
To illustrate, consider a case where we need to detect 1000 QPSK symbols. With QPSK, there are four possible symbols per symbol period. Therefore, detecting the sequence would involve evaluating 4^{1000} combinations, a task that is computationally prohibitive. This brute-force approach, while theoretically feasible, is practically unattainable due to the sheer number of computations involved.

(Refer Slide Time: 02:41)



However, a possible avenue for improvement comes from the observation that both y and S_b are summations of signals accumulated over time. This raises the question: could we adopt an incremental approach, making decisions progressively rather than evaluating all possible combinations at once? This is precisely what we aim to explore in this lecture. We will dissect the problem into two parts to investigate this possibility.

(Refer Slide Time: 06:14)



Let's start by examining the first term, the real part of <y, S_b>. This can be expanded as:

$$\mathrm{R}(\langle y, S_B \rangle) = \sum_k \mathrm{Re}\left(B^*_{(k)} Z_{(k)}\right)$$

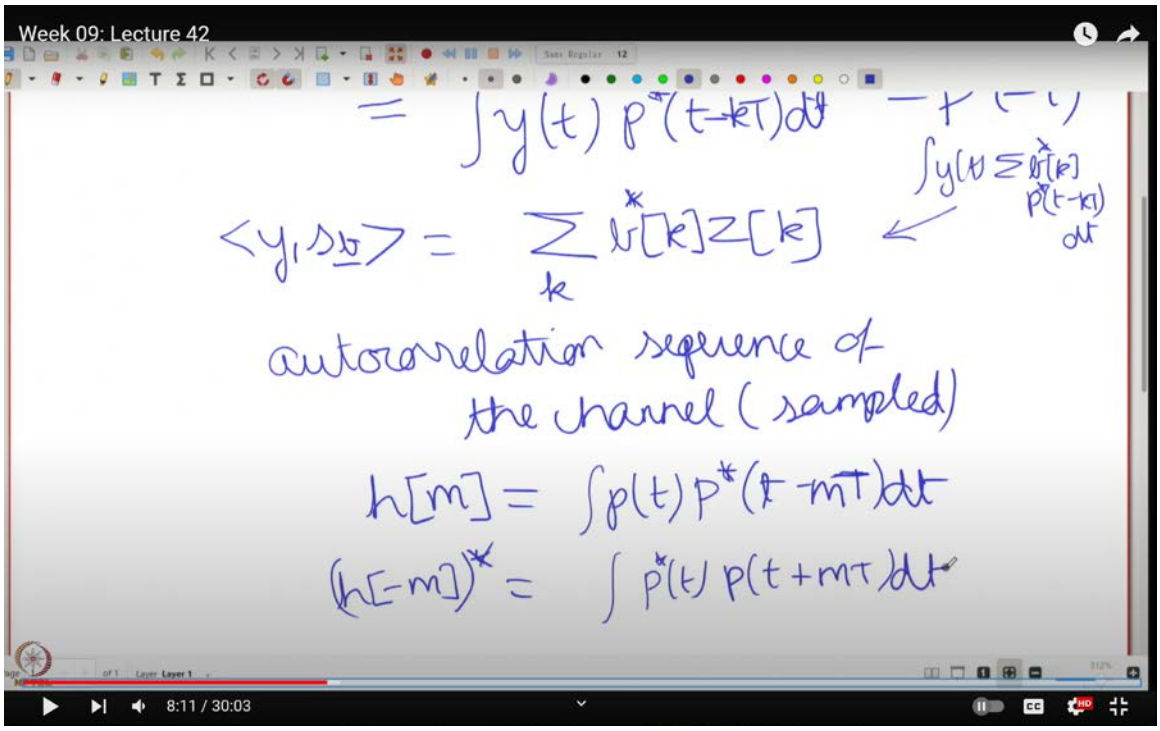
How do we arrive at this? Essentially, it involves performing matched filtering. By applying a matched filter to the observation y, we compute <y, S_b>, yielding Z_n, where $Z_n = y^* \mathrm{P}_{\mathrm{M}f}(kT)$. Let's formalize this step by step.

To revisit the key expression for Λ_B, it is given by:

$$\Lambda_B = \mathrm{R}(\langle y, S_B \rangle) - \frac{|S_B|^2}{2}$$

Recall that the construction of S_b involves a summation over B_k T, accounting for the shifts in time, specifically T - kT.

(Refer Slide Time: 08:11)



hat is essentially what S_b represents. Now, let's define Z_k to correspond to the k-th term. How do we define this? Z_k is given by the matched filter applied to y at kT, i.e., $Z_k = y^* p_{\text{matched filter}}(kT)$. This should be intuitive because <y, S_b> is actually the integral of y(T) and S_b^*(T), and it reduces to the same operation we're describing here. Essentially, you're just taking the k-th sample in this process.

Mathematically, this is expressed as:

$$\langle y, S_B \rangle = \int y(T) S_B^*(T)\ dT$$

Now, if you recall from our earlier discussions, the matched filter $p_{\text{Mf}}(T)$ is the complex conjugate of the original pulse p(-T), so $p_{\text{Mf}}(T) = p^*(-T)$. Given this, when you evaluate the integral, Z_k turns out to be $\int y(T)p^*(T - kT)\ dT$. In simpler terms, Z_k is computed by applying the matched filter, then sampling at discrete intervals.

What's crucial here is that Z_k retains the same meaning as it did when there was no channel involved, it represents the sampled outputs of the matched filter. However, in this scenario, due to the presence of the channel, Z_k carries information not only about the current symbol but potentially about previous symbols as well.

Now, moving forward, let's revisit the expression <y, S_b>. This is given by the summation over k of $B_k^* Z_k$. How did we derive this? It becomes evident when you write out the definition explicitly:

$$\langle y, S_B \rangle = \int y(T) S_B^*(T)\ dT$$

If you decompose the signals y(T) and $S_b(T)$, you will find that they share similar structures. When you break it down, you realize that Z_k essentially handles this summation. That is, Z_k corresponds to the matched filter's output at the sample point T - kT.

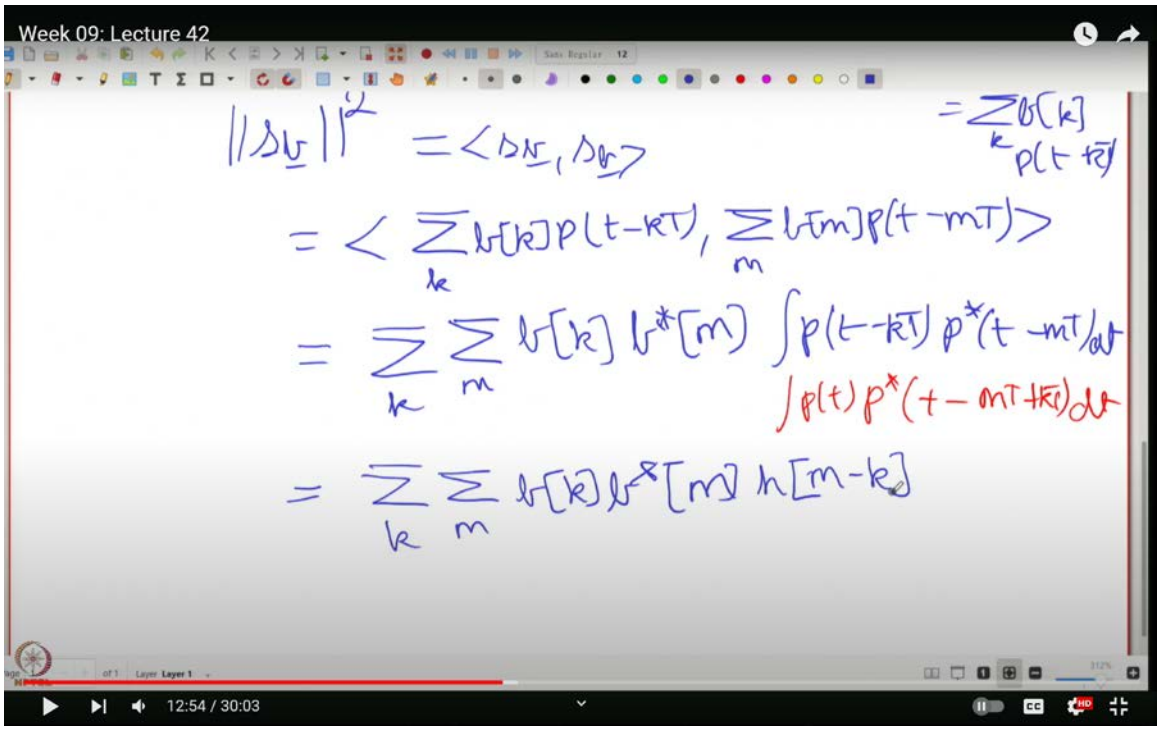
The expression can be rewritten as:

$$\langle y, S_B \rangle = \sum_k B_k^* Z_k$$

where Z_k is formed by integrating y(T) with the matched filter $p^*(T - kT)$. The key insight here is that since S_b is a summation of several shifted pulse shapes, Z_k similarly involves summing over multiple such terms. By taking the summation inside the integral, you ultimately obtain the expression we just defined.

Now, an important point to note is that we're aiming for an incremental detection strategy, one where we can progressively add terms and make decisions about the transmitted symbols. This is the core idea we are building towards.

(Refer Slide Time: 12:54)



Next, we will define the autocorrelation sequence of the channel, specifically for the sampled version of the channel. What does this mean? Think of it this way: when you calculate the magnitude squared of y, you are implicitly convolving it with the channel. If you multiply y by S and integrate, you essentially end up correlating the channel's response with itself. Mathematically, this results in convolving the channel's impulse response h with its time-reversed counterpart, $h^*(-t)$. We refer to this as the autocorrelation sequence.

These samples of the autocorrelation function will prove to be quite valuable as we move forward with our derivations. For now, let's hold onto this concept as it will become clearer later in the process.

Let me now introduce a new definition. I will call it H_m, and it is defined as follows:

$$H_m = \int p(t) \cdot p^*(t - mT)\, dt$$

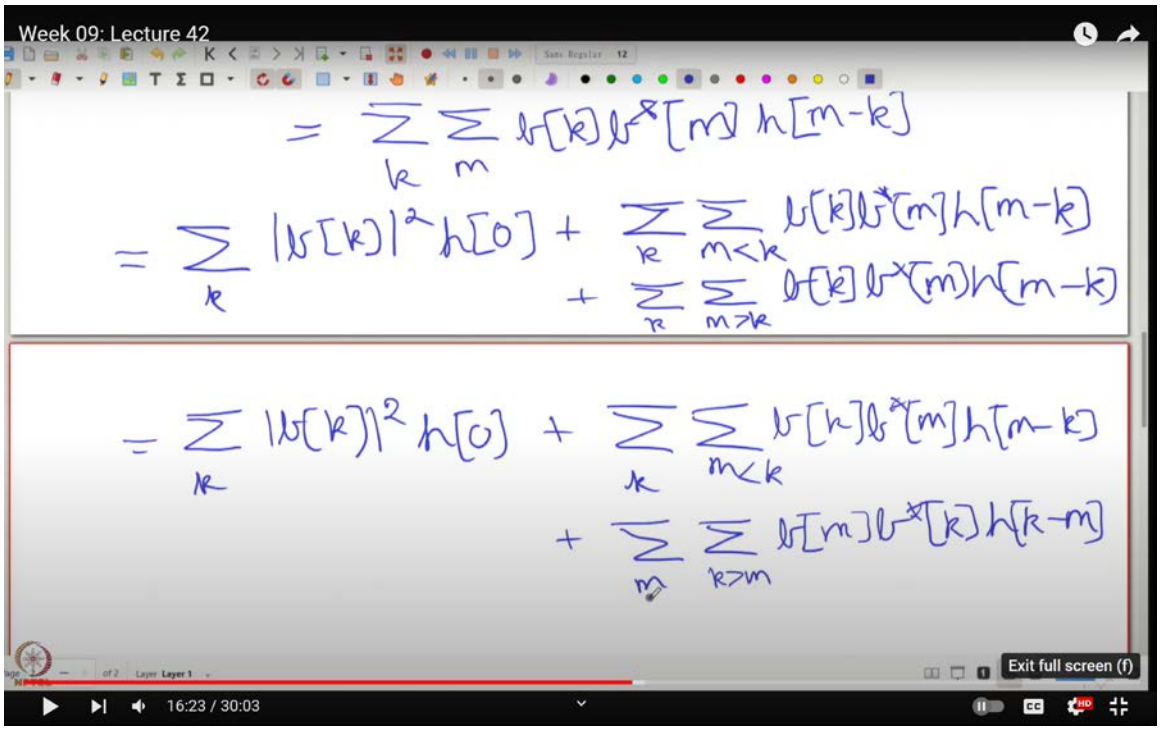
Here, T represents the symbol duration, and H_m essentially forms an autocorrelation sequence. This autocorrelation property holds even for complex channels. Given that H_m is an autocorrelation function, it also possesses the conjugate symmetry property. For instance, $H^*(-m)$ must equal:

$$\int p(t) \cdot p^*(t + mT)\, dt$$

You can see that this integral is computed over t, and by substituting t + mT as a new variable, say u, the expression simplifies to:

$$\int p^*(u - mT) \cdot p(u)\, dt$$

(Refer Slide Time: 16:23)



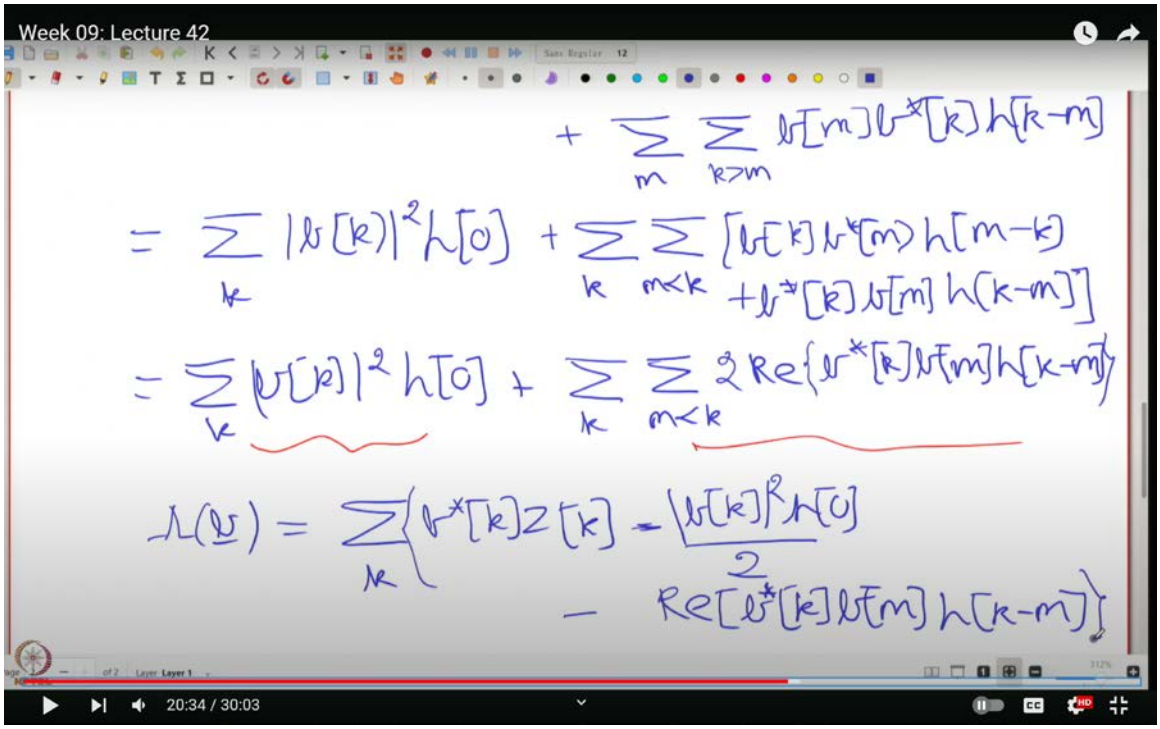
However, this is the same as our original expression for H_m, confirming that the property holds. This definition of H_m, which represents the deterministic autocorrelation of the channel, will prove very useful in our derivations moving forward. It's much easier to work

with than the convolution of p and p^*, sampled at different intervals, which is more complex and challenging to express.

Effectively, H_m represents the convolution of p and p^* but in a shifted form, specifically the convolution of $p^*(-t)$ with p(t). With this autocorrelation sequence established, we can now move on to the next key term, which is $|S_b|^2$, the squared norm of S_b.

Intuitively, $|S_b|^2$ represents the energy of the combined signal, or more formally, the integral of the squared magnitude of the signal. But it's important to remember that this quantity also contains information about the transmitted symbols, so we will need to account for that as we proceed.

(Refer Slide Time: 20:34)



Let's begin with the definition of $|S_b|^2$. Recall that S_b was defined as the summation:

$$S_B = \sum_k B_k \cdot p(t - kT)$$

Thus, we can express the squared norm as:

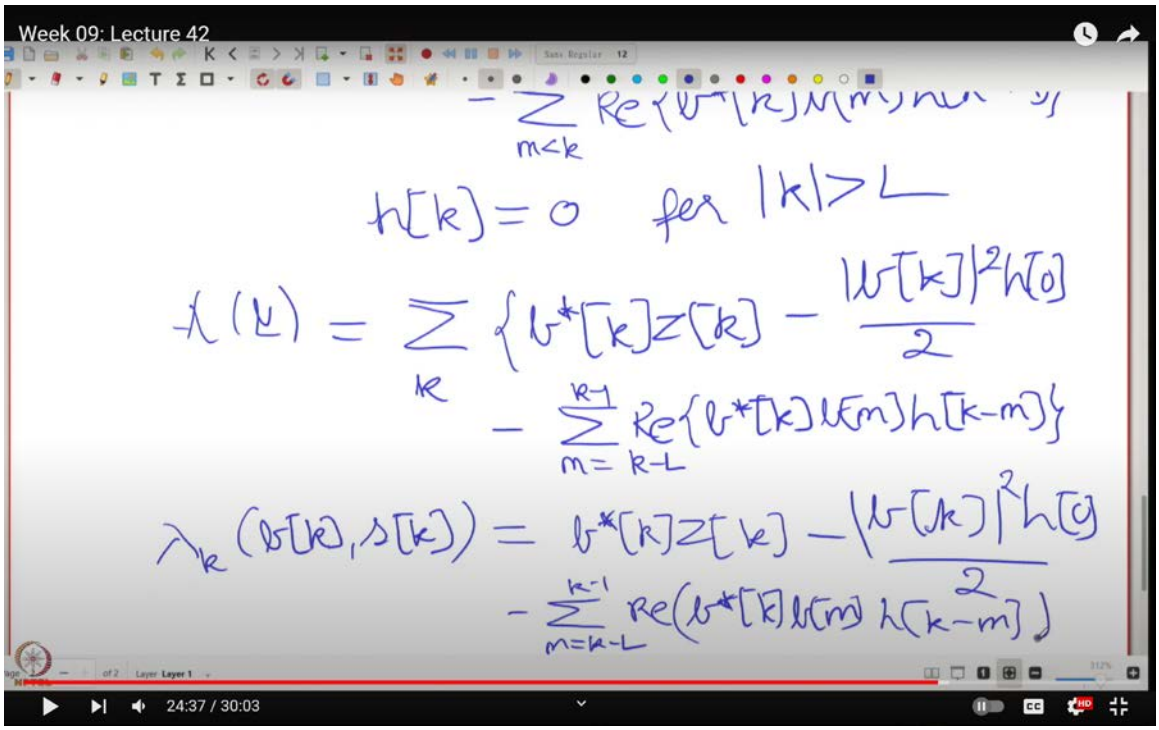
$$|S_B|^2 = \langle S_B, S_B \rangle$$

However, to facilitate some manipulations in the upcoming summation, I'm going to use different indices for the two instances of S_b. So I'll rewrite this as:

$$|S_B|^2 = \sum_k B_k \cdot p(t - kT), \sum_m B_m \cdot p(t - mT)$$

There are no surprises here, I'm merely re-expressing S_b using different indices so we can manipulate the summations more easily. If we were to use the same index for both terms, the double summation could become incorrect or unnecessarily complicated.

(Refer Slide Time: 24:37)



Now, we want to simplify this expression. Notice that the inner product operation is linear, meaning we can rearrange the summation order and the inner products without issue. Thus, we can rewrite it as a double summation:

$$\sum_k \sum_m B_k \cdot B_m^* \int p(t - kT) \cdot p^*(t - mT)\, dt$$

At this point, it's helpful to recall our earlier definition of H_m. If you compare this term with the expression for H_m, you'll notice a close relationship. Specifically, the integral:

$$\int p(t) \cdot p^*(t - mT + kT)\, dt$$

can be transformed into a form resembling H_m by shifting the variables appropriately. This manipulation significantly simplifies the expression, allowing us to represent the result in terms of H_m.

Thus, we can now express the squared norm as:

$$|S_B|^2 = \sum_k \sum_m B_k \cdot B_m^* \cdot H_{m-k}$$

This greatly simplifies the calculations, leveraging the properties of the autocorrelation sequence H_m.

Alright, let's delve into this carefully and explain the reasoning behind this sequence of steps. We'll be performing some strategic manipulations, breaking things down into parts, and then reassembling them in a more intuitive form.

First, let's not get too caught up in the integral form. Instead, let's focus on splitting the summation based on the value of m in relation to k. So, I'm going to rewrite this expression into three distinct parts:

1. The first part corresponds to when m = k. In this case, the expression simplifies to $|b_k|^2 \cdot h(0)$. This is because we're primarily interested in the k-th symbol, and for m = k, the autocorrelation sequence h(m - k) reduces to h(0), which corresponds to the energy contribution of that specific symbol.

2. The second part involves summing over all terms where m < k. Here, the expression becomes $\sum_k \sum_{m<k} b_k \cdot b_m^* \cdot h(m - k)$. This accounts for contributions where the index m is less than k, capturing the interactions of previous symbols with the current one due to the convolutional nature of the channel.

3. The third part deals with the scenario where m > k. In this case, the expression is $\sum_k \sum_{m>k} b_k \cdot b_m^* \cdot h(m-k)$. Here, the contributions come from future symbols influencing the current symbol because of the channel's dispersive characteristics.

So, the overall expression now reads:

$$\sum_k \left(|b_k|^2 \cdot h(0)\right) + \sum_k \sum_{m<k} b_k \cdot b_m^* \cdot h(m-k) + \sum_k \sum_{m>k} b_k \cdot b_m^* \cdot h(m-k)$$

Now, to make this a bit more elegant, let's exploit a small trick: we can swap the roles of k and m in the third term. This allows us to rewrite it as:

$$\sum_m \sum_{k>m} b_m \cdot b_k^* \cdot h(k-m)$$

By doing this, we notice that the second and third terms are now structurally similar, meaning we can combine them.

So, combining these two terms, the expression becomes:

$$\sum_k |b_k|^2 \cdot h(0) + \sum_k \sum_{m<k} \left(b_k \cdot b_m^* \cdot h(m-k) + b_k^* \cdot b_m \cdot h(k-m)\right)$$

Here, the terms within the summation have a nice symmetry: $b_k \cdot b_m^* \cdot h(m-k)\ and\ b_k^* \cdot b_m \cdot h(k-m)$ are essentially mirror images of each other.

Thus, we've effectively restructured the summation, combining related terms to arrive at a cleaner, more intuitive form that captures the interactions between symbols in a dispersive channel. This step simplifies our calculations moving forward, making the entire expression much more manageable.

Now, here's where the interesting trick comes into play. We begin with the summation over k, starting with the term $|b_k|^2 \cdot h(0)$, which accounts for the energy of the current symbol. Then, for the second term, we introduce a double summation over k and m, where m < k. This double summation includes an interaction term that comes with two times the real part of the expression, and here's why: the expression $b_k \cdot b_m^* \cdot h(k-m)$ involves

both the conjugate terms h(k - m) and $h(m-k)^*$, which are complex conjugates of each other. Therefore, we take twice the real part.

To simplify things further, we can write this as $2 \cdot \mathrm{Re}\left(b_k \cdot b_m^* \cdot h(k-m)\right)$, though it's equally valid to express it as $2 \cdot \mathrm{Re}\left(b_k^* \cdot b_m \cdot h(m-k)\right)$. Both forms are equivalent, but for this explanation, I've chosen the former. What's crucial here is that this term encapsulates the interactions due to inter-symbol interference (ISI), which arises because of the channel's presence.

Notice that if the channel didn't cause interference (i.e., if it were ideal), h(k - m) would be non-zero only when k = m. In that case, you'd only have terms involving $|b_k|^2$, and the scenario would reduce to a straightforward AWGN (Additive White Gaussian Noise) detection approach. So, this interference term is what captures the effect of the channel on the symbols.

Now, let's define λ_b, our overall metric, which represents the cost function or metric for the set of symbols b. It's given as the summation over k, but we must recall the earlier expression where we had $\sum b_k^* \cdot z_k$, representing the matched filter outputs. In this case, λ_b is expressed as:

$$\lambda_b = \sum_k \left(b_k^* \cdot z_k - \frac{|b_k|^2 \cdot h(0)}{2} - \sum_{m<k} \mathrm{Re}\left(b_k^* \cdot b_m \cdot h(k-m)\right) \right)$$

Notice that we subtract this inter-symbol interference term because of the negative contribution due to the channel's effect.

Now, importantly, our next assumption simplifies things significantly. In practical systems, we often assume the channel has a finite length, specifically, it behaves like an FIR (Finite Impulse Response) filter. This means that the channel's response h(k) is non-zero only for a finite number of taps, say l taps. So, we assume that h(k) = 0 for all k beyond l - 1. In other words, the channel introduces interference only for the most recent l symbols.

Given this assumption, we can rewrite λ_b more succinctly:

$$\lambda_b = \sum_k \left(b_k^* \cdot z_k - \frac{|b_k|^2 \cdot h(0)}{2} - \sum_{m=k-l}^{k-1} \mathrm{Re}\left(b_k^* \cdot b_m \cdot h(k-m)\right) \right)$$

Here, we observe that the decision on the current symbol b_k is influenced only by the past l symbols due to inter-symbol interference (ISI) introduced by the channel. This reflects the memory effect of the channel, indicating that decisions on b_k are affected by the preceding l symbols. This is the key insight: you only need to consider the last l terms of memory when making a decision on the current symbol, which greatly simplifies the complexity of the problem.

So, now we've been eagerly discussing this concept of an additive metric. But why exactly is this called an additive metric? Well, the reason is that it's expressed as a summation over k, which allows us to proceed incrementally. Each term simply adds to the previous ones, so we can keep building the metric step by step. To express this more formally, let's use some notation. We'll denote it as λ_k and write it as a function of b_k and S_k, where S_k represents a part of the signal vector S_b. The exact notation might not be as critical here; what's important is understanding how the k-th term evolves in this context.

So, this can be written as:

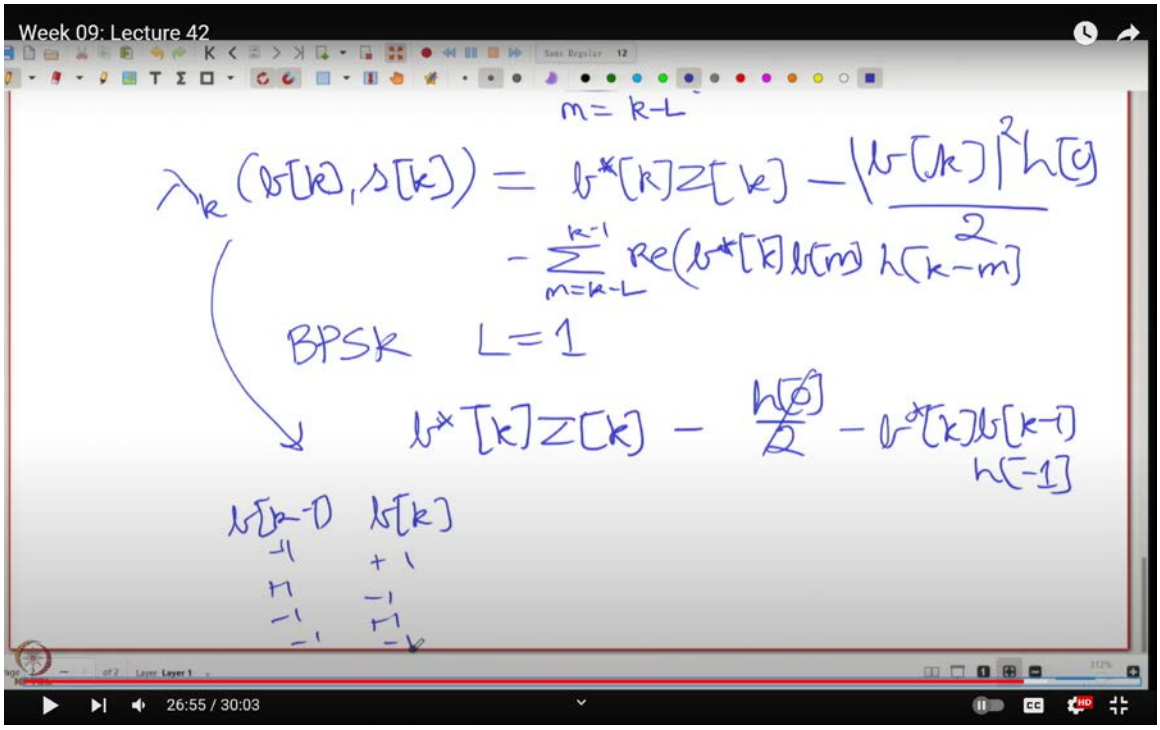
$$\lambda_k = b_k^* \cdot Z_k - \frac{|b_k|^2 \cdot H_0}{2} - \mathrm{Re}\left(b_k^* \cdot b_m \cdot H(k-m)\right),$$

where Z_k represents the received signal, H_0 is the channel gain, and the real part term captures the inter-symbol interference (ISI). I almost forgot the summation symbol here, so let me correct that: there's a summation over the past symbols, typically over m, which accounts for the memory introduced by the channel.

Now, the beauty of this additive metric is that it's simple to compute. The key reason is that you only need to account for the last l symbols due to the memory introduced by the finite impulse response (FIR) of the channel. This significantly reduces the computational burden because you don't need to evaluate all possible symbol combinations across the entire sequence.

However, keep in mind that this additive metric alone doesn't directly tell you the optimal value of b_k. Instead, it provides a framework that helps you combine different possibilities for each symbol. Let me give you a clearer example of how this works.

(Refer Slide Time: 26:55)



Suppose we focus on the past l symbols, specifically $b_{k-1}, b_{k-2}, \ldots, b_{k-l}$. For simplicity, let's assume l = 1. In this case, the metric simplifies, and we only need to consider the immediate past symbol, b_{k-1}. Let's say we are working with binary phase shift keying (BPSK), so b_k can be either +1 or -1.

In this scenario, the additive metric becomes:

$$\lambda_k = b_k^* \cdot Z_k - \frac{H_0}{2} - b_k^* \cdot b_{k-1} \cdot H(-1),$$

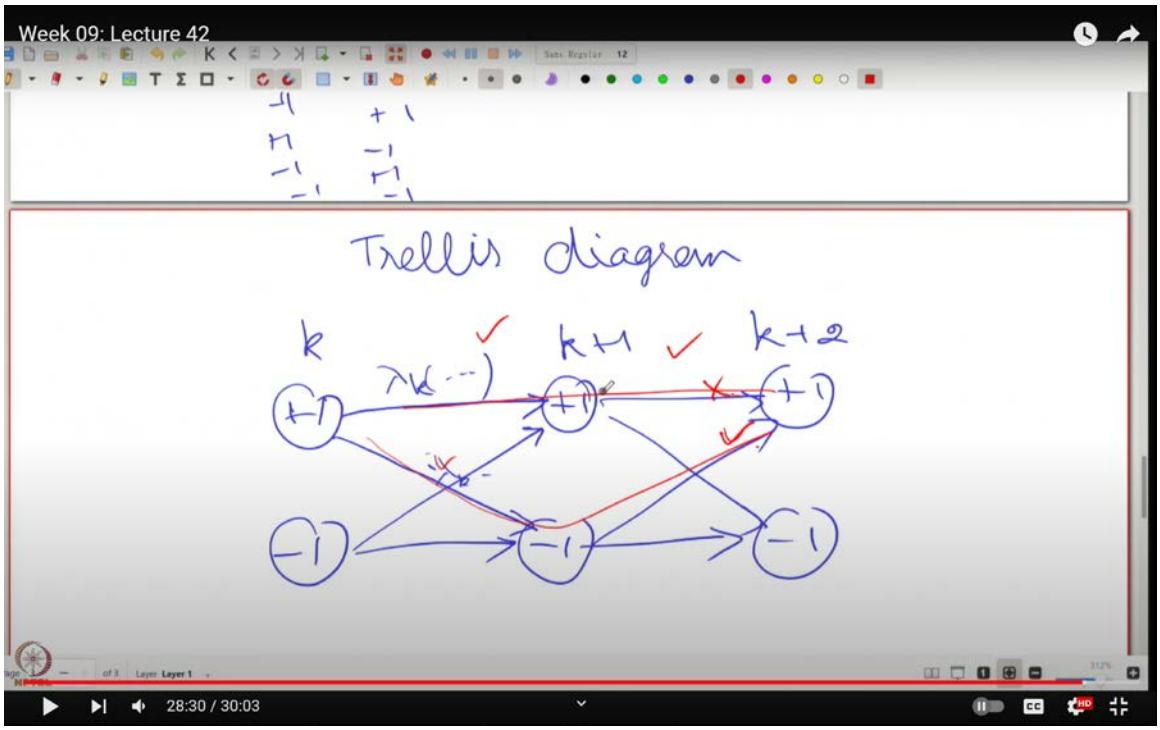
where H(-1) accounts for the inter-symbol interference caused by the previous symbol. Since l = 1, we only have one past term, so there's no need for a summation.

Now, we can evaluate this metric for all possible combinations of b_k and b_{k-1}. For instance, we'll calculate the metric for:

- $b_k = +1$ and $b_{k-1} = +1$,
- $b_k = +1$ and $b_{k-1} = -1$,
- $b_k = -1$ and $b_{k-1} = +1$,
- $b_k = -1$ and $b_{k-1} = -1$.

We evaluate the metric for each of these combinations, and this process continues incrementally as we move forward through the sequence. After evaluating b_k, we move on to b_{k+1}, then b_{k+2}, and so forth. Each evaluation builds upon the previous one, incorporating the effects of inter-symbol interference, until we reach a point where we can make reliable decisions on the past symbols.

(Refer Slide Time: 28:30)



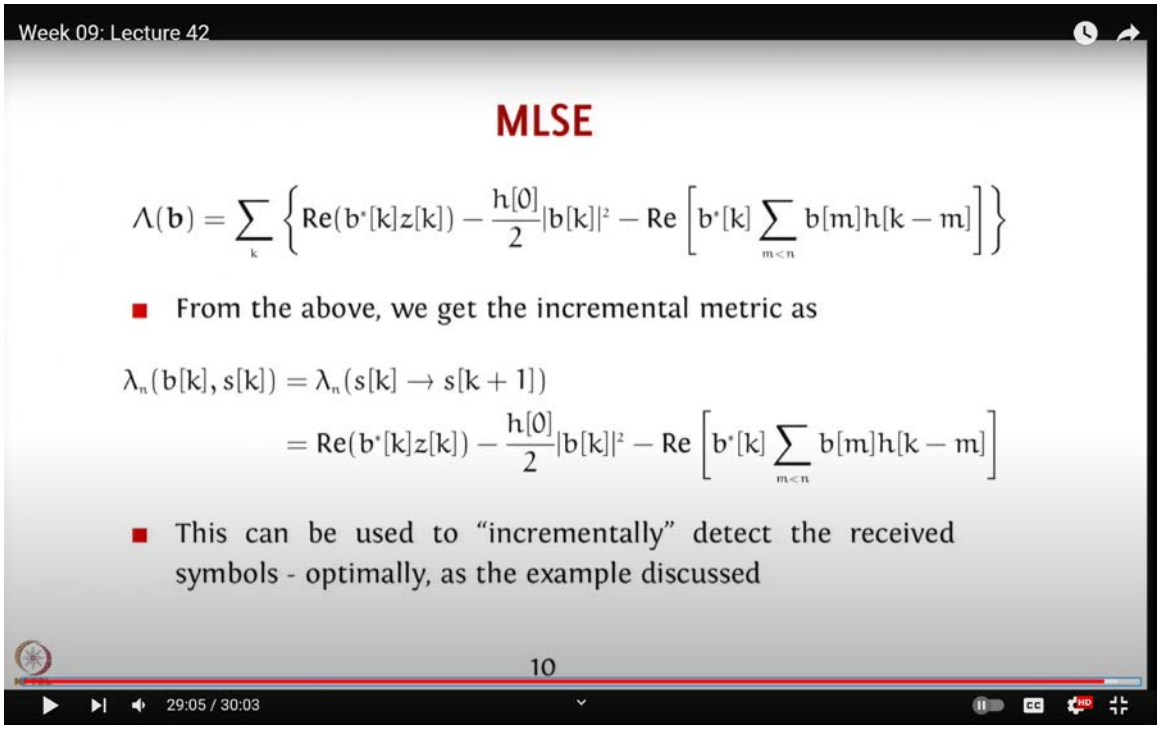
This approach ensures that the complexity of detecting the symbols remains manageable, even in the presence of channel interference, and leverages the structure of the problem to achieve optimal decisions without evaluating every possible sequence of symbols.

The question we now face is: can this approach actually be implemented effectively? This is precisely where the Viterbi algorithm comes into play. The idea is to construct what is known as a trellis diagram. Let's assume that at time k, the symbol b_k can take on a value of either +1 or -1. Similarly, at time k + 1, b_{k+1} can be either +1 or -1, and the same applies at time k + 2, and so on.

What we're going to do is evaluate the metric λ_k for each possible transition:

- Evaluate the metric for a transition from +1 to +1,
- Evaluate the metric for a transition from +1 to -1,
- Evaluate the metric for a transition from -1 to +1,
- Evaluate the metric for a transition from -1 to -1.

(Refer Slide Time: 29:05)



We'll continue this process step by step. The key point here is that these metrics are additive, this is because we're dealing with a norm squared and a summation over k. So, we can start adding the metrics at each point: add the metric for this path, add the metric

for that path, and so forth. As we move forward, these paths will correspond to different symbol sequences.

Now, for each of the possible paths in the trellis diagram, we will evaluate the total metric. Suppose you have two paths leading to a particular point: one will have a higher metric and the other a lower one, assuming we want to maximize the metric. Naturally, the path with the higher metric will be the one that survives, and the path with the lower metric will be discarded. This is the crux of the Viterbi algorithm, at each step, we eliminate less likely paths and retain only the optimal ones.

By following this approach through the trellis diagram, we significantly reduce the complexity of the search. Instead of having to evaluate all possible combinations of symbols, which would involve an unmanageable number of searches, potentially something like 4^{1000} searches, we use the trellis and the Viterbi algorithm to make decisions more efficiently.

This is where the Viterbi algorithm becomes incredibly useful. We'll delve into the details of the Viterbi algorithm in the next lecture. To summarize what we've covered so far, we took the real part of the inner product $< y, S_b >$ and the norm $|S_b|^2$ and broke them into more manageable components. Using these components, we derived an additive metric, which allows us to incrementally detect the received symbols in an optimal manner.

It may not be immediately obvious how this all comes together, but in the next lecture, we'll go over the Viterbi algorithm in detail, along with an example that will make everything clearer. We'll continue with the example problem we started discussing earlier. That's all for now. Thank you.