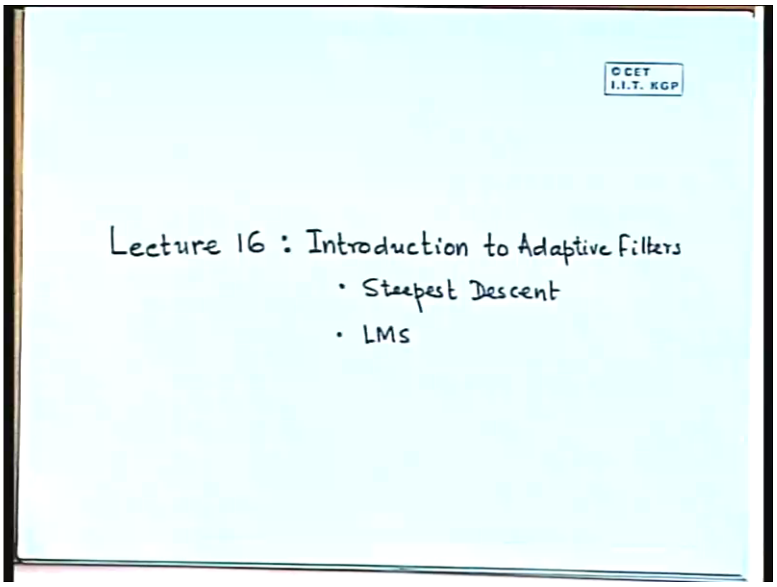
Estimation of Signals and Systems

Prof. S. Mukhopadhyay
Department of Electrical Engineering
Indian Institute of Technology, Kharagpur

Lecture - 16
Introduction to Adaptive Filters – Steepest Descent and LMS

So, good morning we are having our last class before mid-semester and that is why we will finish the topic of filters today.

(Refer Slide Time: 0:58)


CCET
I.I.T. KGP

Lecture 16 : Introduction to Adaptive Filters
- Steepest Descent
- LMS

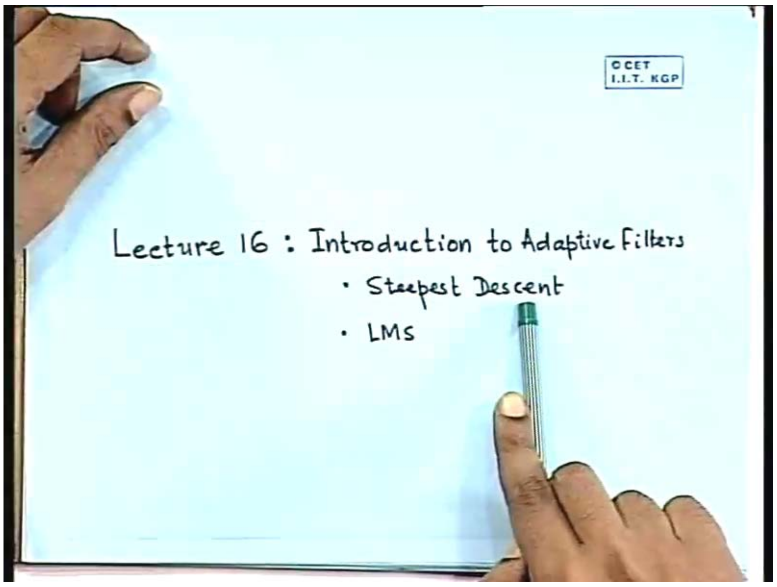
Filters means, input output filters and take up new topics after the mid semester. So today we will talk about, adaptive filters little bit. That is those filters that we have, what sort of a problem have we seen before? We have seen that, if we take a filter structure we have we have mostly worked with FIR structure; we have seen that given the statistics of the data, how to calculate weights in that is the coefficients of the FIR filter, such that you get minimum mean square error, this problem we have solved.

Now in practise there are there are several reasons, why in practise adaptive filters are very well used. So adaptive filters are filters; where you do not you know is not that, you are going to take a lot of effort and then calculate the noise statistics nicely and then you will calculate one set of weights and, then freeze them, is not going to, that usual does not work. So what works is that you have that filter structure and then you keep on changing the weights.., I mean you you keep on kind of sensing the noise statistics, all the time. And depending on the kind of noise statistics that you see, you you you keep on changing the weights, right. So you have an element of adaptation in these filters, okay.

So so so we are going to, what we are going to see today is.. let people have let people sit down, then we will continue. So so what we at going to see today is that is, what are the basic ideas of adapting, you know adaptation is a is a is a is a rather complex business; I mean is not whenever you have an adaptive system its behaviour, to analyse its behaviour is is not trivial, it involve various things quite complicated but we will not going to all that. We will just, we will do two things; firstly we will try to identify some basic rational for for adaptation, that is if we want to change weights, what should be our basic guiding principles?

And then we will see two major two major algorithms which are quite used and see how they satisfy those principles and try to ensure that, they have you know good properties, other than that we are not going to do anything. We you you you can have I mean whole volumes and hundreds of papers written on adaptive filters, but we are not going to do all that we are going to look at it in one class. So we will basically look at two major algorithms; one is called steepest descent and the other is called LMS.

(Refer Slide Time: 03:50)

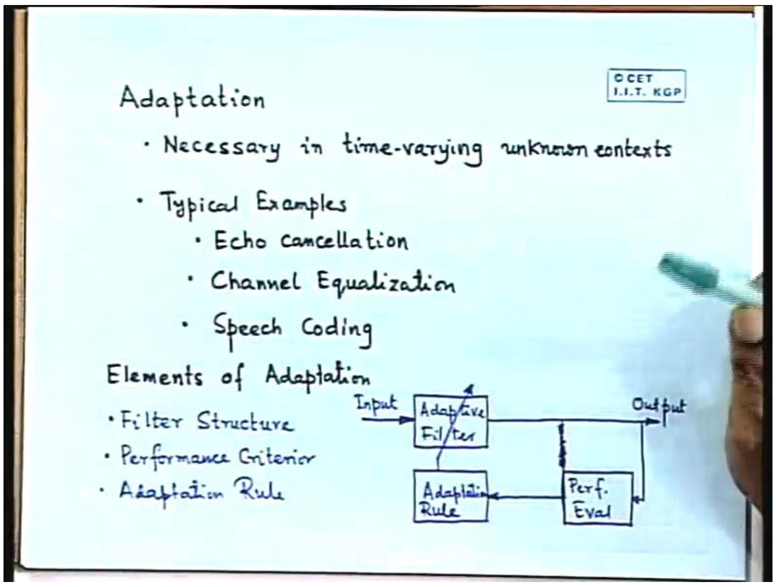

Lecture 16 : Introduction to Adaptive Filters

- Steepest Descent
- LMS

CCET
I.I.T. KGP

There are various other versions of adaptive filters which we are not going to consider in this course.

(Refer slide Time: 04:02)



Adaptation

- Necessary in time-varying unknown contexts
- Typical Examples
  - Echo cancellation
  - Channel Equalization
  - Speech Coding

Elements of Adaptation

- Filter Structure
- Performance Criterion
- Adaptation Rule

OCET
I.I.T. KGP
Input
Adaptive Filter
Output
Adaptation Rule
Perf. Eval

So to begin with anyway you have you have adaptation, it it generally happens that; you have something which is which is either unknown or time varying. So you have so when when something is unknown, what you do is you start with some guess and then we, you you keep on improving you guess based on some some criterion. So you you you adapt your guess and hopefully if you are if if your adaptation rule is good enough, you will come fairly close to the real good guess right, which is may ne may be which is the true value.

It sometimes happens that, the situation changes, that is suppose; now you have one kind of noise statistics, after two seconds you have another kind of noise statistics. So is not possible to have a constant set of you know optimal wiener filter coefficients, because things are going to change and and you cannot pre-compute them. So in such a situation you you also need to have a scheme which will continuously track these changes in the data and will always produce coefficients which are going to be good if not optimal.

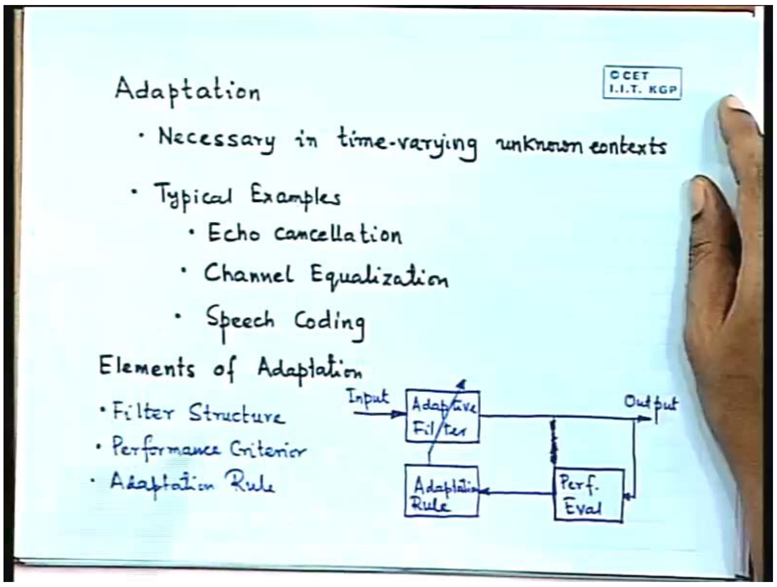
So they will if things do not change they will slowly approach optimal, if things change they will at least track the change, right. This is the this is the best that you can hope for. So so in such situations you need adaptive algorithms. And typical cases where these algorithms are used, this adaptive filters are in you know echo cancellation; I mean have you seen that, especially if you make if you ever happen to talk in overseas calls, telephone calls you can actually make out that, there is a that there is a propagation delay, right. And sometimes there is I mean there because of this propagation delay, sometimes they are tends to be echoes because the transmitter and the receiver are not fully decoupled. So there is so when the when the when the the signal which comes, to some extent it also gets into the receiver and then it is fed back to you, right.

So so so so you tend to get an echo. So it happens that, most telecommunication equipment have what is known as echo cancellers. So they and and these these these echoes depend on the kind of you know, the kind of circuit which is downstream of the point where you put the echo canceller and that can change, because the because the telephone exchange configuration is very dynamic. Its its continuously switch set of systems, so you do not know what is the exact properties of system, which is downstream at any given point of time. So you need to adapt, right.

Similarly; you can have things like, you know channel equalisation that is where you where you try to correct for in the receive signal. You try to correct for accumulations and phase lags, which have occur in the channel and channels can be dynamically changing. For example a a a radio channel can can can change very much depending on you know weather and and other things. So so so channel characteristics, may change right. So you so whenever you you need to equalise something, you you need to have it as adaptive.

And then for also for speech coding, because you know speech coding filters also need to need to changed depending on who is speaking, various kinds of you know I mean frequencies spectrum et cetera, everything will change depending on the person who is speaking at any given point of time. So so even you have an an an adaptive coding schemes. So there are various kinds of I mean, I just wanted to say you that, there are various kinds of applications where this adaptive filters are used. Basically in in in any adaptive systems in that includes, an adaptive filter the basic structure is like this.

(Refer Slide Time: 08:08)



Adaptation

- Necessary in time-varying unknown contexts
- Typical Examples
  - Echo cancellation
  - Channel Equalization
  - Speech Coding

Elements of Adaptation

- Filter Structure
- Performance Criterion
- Adaptation Rule

© CET I.I.T. KGP
Input
Adaptive Filter
Output
Adaptation Rule
Perf. Eval

You know you have a filter; this is the filter which you want to adapt, so that you get some good performance in the output. If you have… in the case of signal processing, this is the filter you could have an adaptive control, you could have so many other things. So now what happens is that this output, the properties of this output are evaluated. That is this is some performance evaluation block. So you have to define some performance criterion typically in terms of the outputs.

So that you can evaluate whether this output is good enough or not, right. And this performance evaluation function generally gives some quantity, which is used by the adaptation rule. So so so the so this adaptation rule is a rule which changes the coefficients in this filter, such that the such that you you get, such an output for which the performance criterion is minimized. So you should always try to minimize some performance criterion and you continuously, try to change your weights such that always this this this performance criterion when when when evaluated has to be seen as small as possible.

So this is the structure of most adaptive systems, I mean otherwise how do you adopt, right.

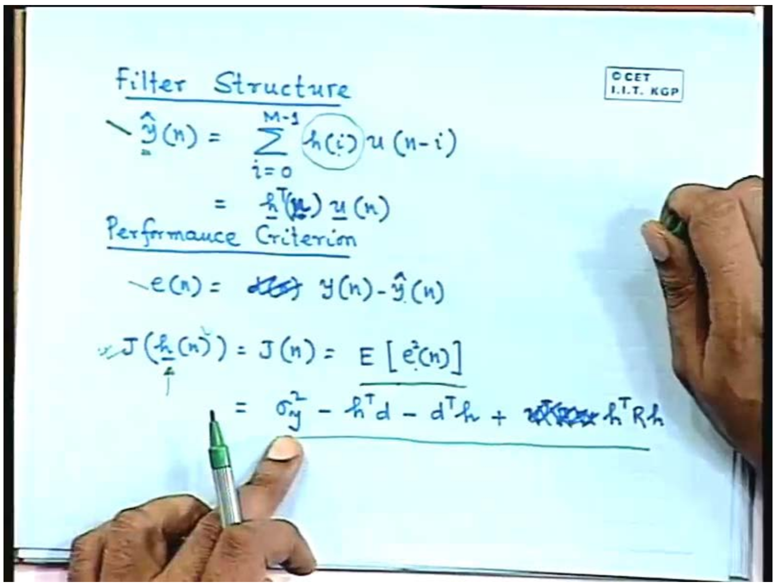
So you have three major quantities, you have you have a filter structure; this one in which you change the coefficients, so you have to define that. You have to define a performance criterion; some may be some noise, query or whatever you decide. And you had and you need to have an adaptation rule. So whenever you want to define an adaptive system; these three quantities have to be defined you have to defined a filter structure, you have to define a performance criterion, and you have to define an adaptation rule. Then you have the then if you put them together, you have an adaptive system, right. So now,

Student< is the performance criteria and adaptive rules are fixed>

Yes yes .You know you can have various levels of adaptation, right. For that matter, control is also a kind of adaptation in the sense that, you do not apply a constant signal. It is an adaptation, at the signal level. So the signal changes signal magnitude changes depending on the error. Then you then at a at a higher level of adaptation, you actually try to change the controller coefficients, that is what we call adaptation.

Now now if you want to change any one of these dynamically, then you are trying to change the adaptation rule. You are trying to change an adaptive system itself, such systems are generally called learning systems. So learning systems is a higher level of adaptation where you try to change the adaptation system itself, which changes the controller in turn, which changes the signal in turn, right. So you can have a various levels of things, but this is what is generally understood as a as adaptation, so in our case we have that, well known filter structure which we are very familiar with.

(Refer Slide Time: 11:17)



Filter Structure

$$\hat{\underline{y}}(n) = \sum_{i=0}^{M-1} h(i)\, u(n-i)$$

$$= \underline{h}^T(n)\, \underline{u}(n)$$

Performance Criterion

$$e(n) = y(n) - \hat{y}(n)$$

$$J(\underline{h}(n)) = J(n) = \underline{E[e^2(n)]}$$

$$= \sigma_y^2 - h^T d - d^T h + h^T R h$$

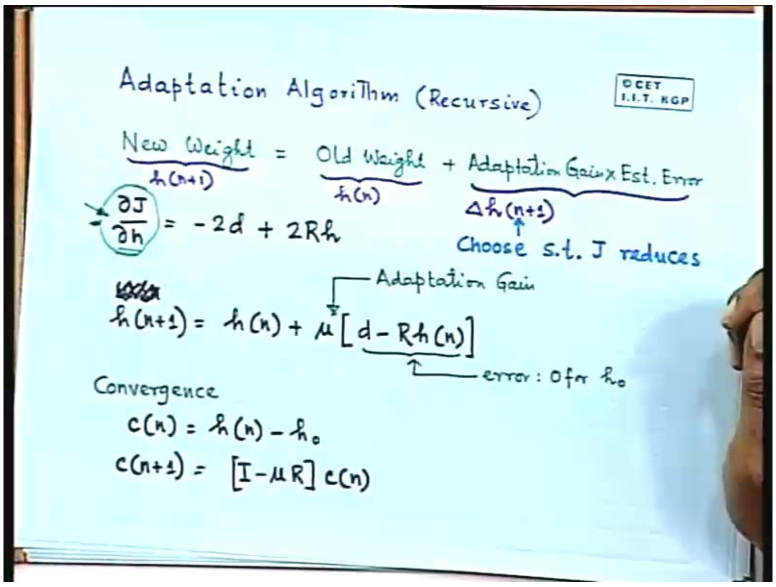
So we are still having trying to predict from samples of u; by an FIR structure some signal y. This this what we know we we have tried to see and we have tried to we we have seen, what are optimal estimates for these, etcetera its it. So this is the filter structure which have defined that, we are going to consider. What is our performance criterion? Performance criterion is this. So we have for the these are the things, that we are going to adapt; such that this is minimised. So obviously the performance criteria depend on E n, which depends on y hat n, which depends of h n. So therefore the performance criteria depend on h n.

So if we change this if we change this, this will change. So our our our objective is to change these, adapt these which are the coefficients of the filter, such that this quantity is minimised, right. This is our this is our intention. And this we must be able to do all the time, even if we know the signal properties, d and R change with time, we should we should we should make corresponding changes in our estimate automatically that is the problem.

So we we are we are familiar with this formula; this is this standard this y n, if you if in place of y hat n, you put these and then do e transpose, rather e transpose e, then you get this one. This we have done in last class also. And is the this is the well-known formula, right. This is simply y minus h transpose u, whole transpose y minus h transpose u, if you just cross multiply you will get this term, is clear now? So we have this filter structure and we have this performance criterion.

Now the main thing is to is to design an adaption law. So we have indicated what we want to change and we have indicated our wish, that we want to minimise, this this is our wish. Now we have to answer the crucial question, as to how to satisfy our wish.

(Refer Slide Time: 13:46)



Adaptation Algorithm (Recursive)

OCET I.I.T. KGP

$$\underbrace{\text{New Weight}}_{h(n+1)} = \underbrace{\text{Old Weight}}_{h(n)} + \underbrace{\text{Adaptation Gain} \times \text{Est. Error}}_{\Delta h(n+1)}$$

Choose s.t. J reduces

$$\boxed{\frac{\partial J}{\partial h}} = -2d + 2Rh$$

Adaptation Gain

$$h(n+1) = h(n) + \mu\left[\underbrace{d - Rh(n)}\right]$$

error : 0 for h_0

Convergence

$$c(n) = h(n) - h_0$$

$$c(n+1) = [I - \mu R]\, c(n)$$

So typical adaptive algorithms are you know; there are two guiding principles, you you will find that ninety-five per cent, probably ninety-nine per cent of of all adaptive algorithms come in this form. That is you want to whatever you want to adopt, say in this case you want to adapt h n, right. So the new new weight h n plus 1, you are continuously as you are getting new data, you are adapting the weight. So for every data sample, you are getting a.. you are you are computing a new weight.

So it always turns out that, they that such a such a form such an adaptation law results. This is this you will find in adaptive control; there are hundreds of adaptive control laws. Similarly you know in all adaptive filter laws most, I mean in I mean in almost all adaptive systems, adaptation laws come in this form. So you have you have the new weight, which you compute in terms, this is your old weight and you want to update.

So always, you tried to compute some form of an error, that is something that would have happened which is which is which would be zero, if the weights were really good because if the weights were really good, then that error will be zero and then there will be there will be no updating. So you see, if the weights are really good you you you you do not want to change it. You want to change it only when the weights are not good.

So you you are always compute whether the weights are really good in terms of some kind of an error, these I mean all algorithms vary in defining this error, defining this gain. So all variations come because of these two definitions; otherwise they are generally in this form. So you will define some error and you will define and then based on that error, you mean you are going to change it using some gain. So these two remain to be determined, but this is the structure in which you always adapt your system

Now the question is and there is there is also another very very very sound principle; that is after all you want to what you want to do you want to minimize J, right. Your your your your objective is to minimise that J function which you have defined. So if you want to minimize that, then in then in which direction you should change h? You should change it in the direction in which J is, J is decreasing is it not?

You should change h, say J changes with h. So there is a there is a rate of change of J with h. Now suppose whatever is the; so you should always change you should always change h in such a direction, that J reduces this is of this is a I mean sound principle, intuitively, okay. So so how do you find that how do you find that direction. So it is says that, you calculate this one ... there is a big manufacturing defect in this pens. So what is done is you calculate delta J, delta h. So what is delta J, delta h? Delta J delta h says that, if you change h to h plus delta h, then then then then J changes to J plus delta J. So you should you should always update in the direction of minus delta J, delta h. This is the, why?

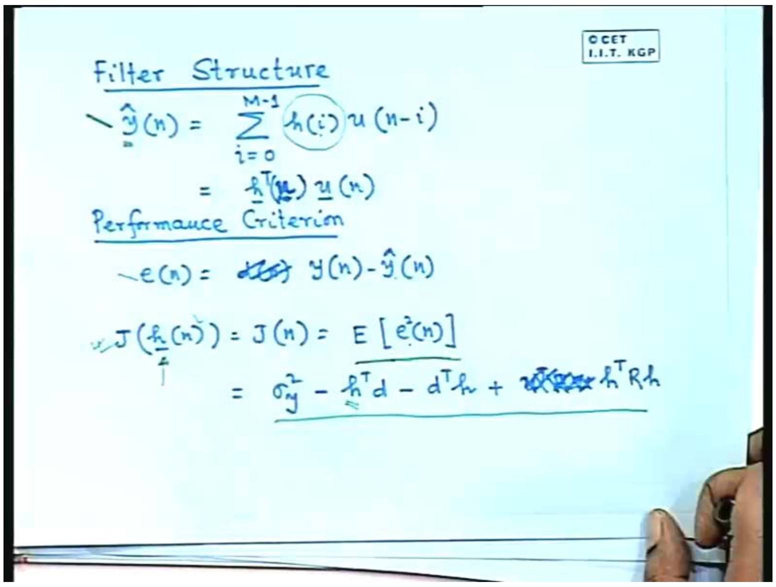
Student< because its tendency is other side>

Yes because if if there is if you change h to h plus delta J, h plus delta delta even. J will change to J plus delta J, you want J to reduce. So you so you make it change towards minus delta J, right.

So you always change. So you so this is the most of the I mean a large number of adaptive algorithms are actually gradient algorithms, where you the the the update is actually based on the gradient of the performance function. For example, all neural networks training algorithms, most neural network trading algorithms are gradient algorithms; that is also an adaptation, you are you you are adapting weights to to to match something. It is exactly this problem, it is a exactly the adaptive filtering problem, training of network. So so everywhere so this is the standard rule, that you update your weights in the negative of the gradient direction, right. So that you are performance criterion is always decreasing at a at every point it is decreasing, right.

So here also so so we found now now, let us calculate delta J, delta h from where, from this formula. We know already J in terms of h, here here you already have J in terms of h.

(Refer Slide Time: 18:44)



©CET
I.I.T. KGP

Filter Structure

$$\hat{\underline{y}}(n) = \sum_{i=0}^{M-1} \boxed{h(i)}\, u(n-i)$$

$$= \underline{h}^T(n)\, \underline{u}(n)$$

Performance Criterion

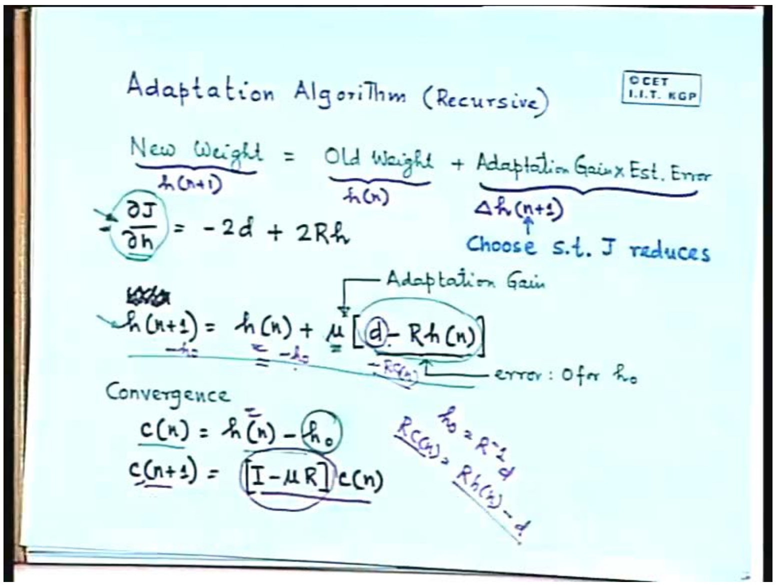
$$e(n) = y(n) - \hat{y}(n)$$

$$J(\underline{h}(n)) = J(n) = \underline{E[e^2(n)]}$$

$$= \underline{\sigma_y^2 - \underline{h}^T d - d^T h + h^T R h}$$

So we find delta J, delta h from here. That is that is very easy to find, this term will give you a d, this term will give you a d and this term will give you two R h, right. So you get minus two d plus two R h.

(Refer Slide Time: 19:04)



Adaptation Algorithm (Recursive)

OCET I.I.T. KGP

$$\underbrace{\text{New Weight}}_{h(n+1)} = \underbrace{\text{Old Weight}}_{h(n)} + \underbrace{\text{Adaptation Gain} \times \text{Est. Error}}_{\Delta h(n+1)}$$

Choose s.t. J reduces

$$-\frac{\partial J}{\partial h} = -2d + 2Rh$$

Adaptation Gain

$$h(n+1) = h(n) + \mu\,[d - Rh(n)]$$

error : 0 for h_0

Convergence

$$c(n) = \bar{h}(n) - h_0$$

$$c(n+1) = [I - \mu R]\, c(n)$$

$h_0 = R^{-1} d$

$Rc(n) = Rh(n) - d$

So delta J, delta h is minus two d, plus two R h. So what is going to be your now now you have to do it, this is your adaptation gain and this is your minus delta J, delta h? I mean there is a factor of half which can be observed here. The factor of two is is not a problem, you have to define this at that the factor of two is observed, okay.

So now so this is now your adaptation rule which ensures that, at every point the gradient is decreasing, right. Now now how do you characterise that, now the move like to know or typically you would like to analyse that how fast is is decreasing, this h is actually a vector. So it has various components h one, h two, h three. So all of them are supposed to be decreasing, when I coming to the coming to the true value; first of all are they coming to the true value, what is the true value? True value is the optimal value, which you would have got suppose; R is not changing then then we know, that if we had calculated the wiener filtering solution, we would got the best possible ever may see, not better is possible.

So, now if we start from some arbitrary initial guess; say h zero and if we apply this adaptation law, do we come to that optimal weight? If we if we come to that, then it is then it is good in a sense; it is at least at least approaching the best. So does that happen and if it happens, how fast does it happen and when does it happen? So to so to analyse that, let us assume that this is an error I have defined; so this is this this is the optimal weight h zero, so this is my error in estimation at any time n, right. This is my this is my estimate according to this law and this is the best possible, so so this is the this is the error.

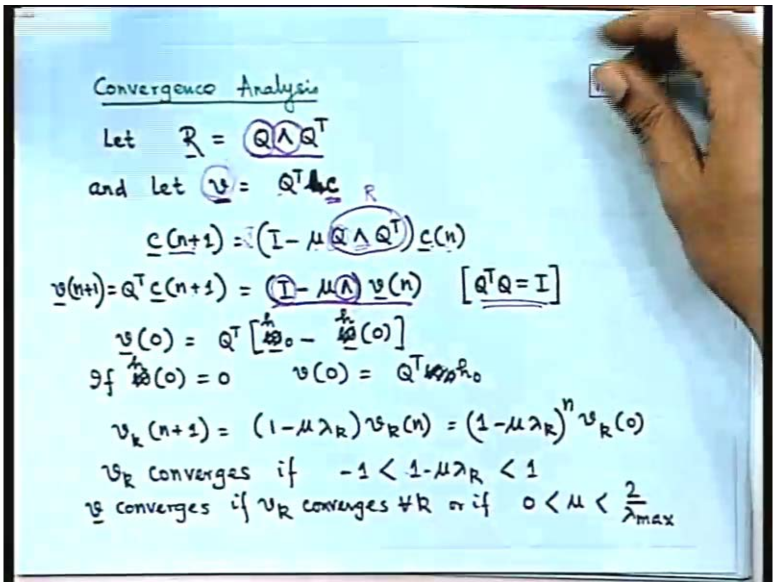
Now what will happen? Similarly, if I now if I now calculate c n plus 1 which is nothing but h n plus 1 minus h 0, that is I had this much of error at instant n. Now according to the update law if I calculate h n plus one, I will have this much of error at n plus one has the error decreased. That is the question that we are asking and eventually at least given enough time, will the error go to zero? This is the question that we are asking, okay.

So for that for doing that basically you have to you have to you have to set up a a recursive kind of you know difference equation, x k plus one equal to something into x k; that that sort of an equation and then see whether it is stable, goes to zero or not. So you have to do that, so that is why I am trying to relate. Now you can easily find out that, c n plus1 can be written like this, why? You put minus h 0 here, put minus h 0 here. So this is c n plus 1, this is c n and here, this is okay. So then how do you get one minus mu R c n? So you get the here, see this term you have got c n plus 1 I into c n is c n that also you have got.

So how does this become mu R c n? Because, what is h 0? h 0 is R inverse d, that is optimal estimate, optimal wiener filter estimate. So h 0 is equal to R inverse d; that we already know. So what is now now now, what is R c n? It is equal to R h n minus R R inverse d, which is d. So now so not now if you do, what is this? This is d. So what is a so what is this? This is nothing but minus R c n, correct. R c n is R h n minus d, this is d minus R h n. that is minus R c n, so it c n into I minus mu R c n, correct. Simple, but what we have done is that, now we have set up an error equation, if we are we are now only in talking in terms of errors.

So if the error was here at the here at the nth state, at the n plus 1th step, it will be related like by this. Now whether the error will go to zero or not will totally depend on the property of this matrix, right. So the the easiest way to analyse it is as we know by doing, what is known as Eigen decomposition. Then this matrix will actually become decoupled, that is the we we should see it in a plane which is called the principle axis, where the fall of each error is is actually independent of the fall of the other. So we will get basically we will get a set of set of scalar equations. So we know that, if we we can we can easily achieve it if we try break up, that is the set up a coordinate I mean along the Eigenvectors. So we write that R is equal to this, where this is Eigenvectors matrix and this is the diagonal matrix consisting of Eigen values, remember we have done this before and and then defined a new set of error coordinates.

(Refer slide Time: 24:50)



Convergence Analysis

Let $\underline{R} = \underline{Q \Lambda Q^T}$

and let $\underline{v} = Q^T \underline{c}$

$\underline{c}(n+1) = (I - \mu \overset{R}{Q \Lambda Q^T}) \underline{c}(n)$

$\underline{v}(n+1) = Q^T \underline{c}(n+1) = \underline{(I - \mu \Lambda) \underline{v}(n)}$ $[\underline{Q^T Q = I}]$

$\underline{v}(0) = Q^T [\underline{h}_0 - \underline{\hat{h}}(0)]$

If $\underline{\hat{h}}(0) = 0$ $v(0) = Q^T h_0$

$v_k(n+1) = (1 - \mu \lambda_k) v_k(n) = (1 - \mu \lambda_k)^n v_k(0)$

v_k converges if $-1 < 1 - \mu \lambda_k < 1$

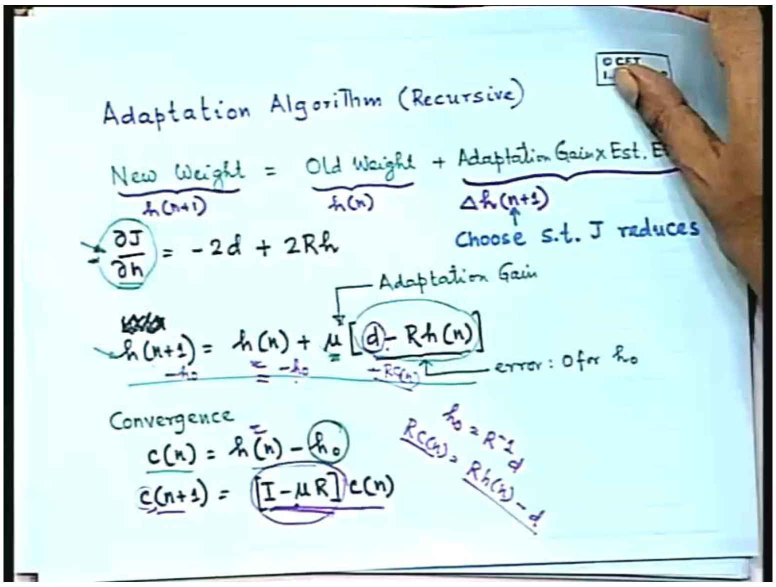
$\underline{v}$ converges if v_k converges $\forall k$ or if $0 < \mu < \frac{2}{\lambda_{max}}$

Now previous error coordinates were c. So now I am trying to define a new error which is v which is defined as Q transpose c. What is the advantage? Advantage is that, if we just take it through this, that that is if you define this then now you just blindly substitute, so c n plus 1 is equal to I minus this is R. So we we are first first substituted this, then then then try to rather than that trying to write in terms of c n plus 1, let us see how this changes? So v n plus 1 is nothing but Q transpose c n plus 1. So you multiply this by Q transpose.

This equation multiply by Q transpose here, so you get Q transpose here, mu is the scalar. So so Q transpose Q is actually I, that also we have proved, that the Eigenvector unitary Eigenvector matrix is its its transpose is its unit is its inverse, if you recall. So if you multiply this by Q transpose Q, this will go to I; there is a gamma here and then Q transpose c, so so you have Q transpose c here which is v n and here also you have Q transpose c which is v n. So you basically get this one, what is the advantage? Why did we do this because this matrix is now diagonal. This matrix is also diagonal, so this whole matrix is diagonal which means that, now each coefficient of v has a is changing by itself, it is not affected by any other any other coefficient.

See see the that this not that does not happen here, this is a non-diagonal matrix. So then c n plus 1 will actually be related to c one, c two, c three many c's at n.

(Refer Slide Time: 26:36)



Adaptation Algorithm (Recursive)

$$\underbrace{\text{New Weight}}_{h(n+1)} = \underbrace{\text{Old Weight}}_{h(n)} + \underbrace{\text{Adaptation Gain} \times \text{Est. E}}_{\Delta h(n+1)}$$

Choose s.t. J reduces

$$\frac{\partial J}{\partial h} = -2d + 2Rh$$

Adaptation Gain

$$h(n+1) = h(n) + \mu\left[d - Rh(n)\right]$$

error : 0 for h_0

$-h_0 \quad -h_0 \quad -Rc(n)$

Convergence

$$c(n) = h(n) - h_0$$

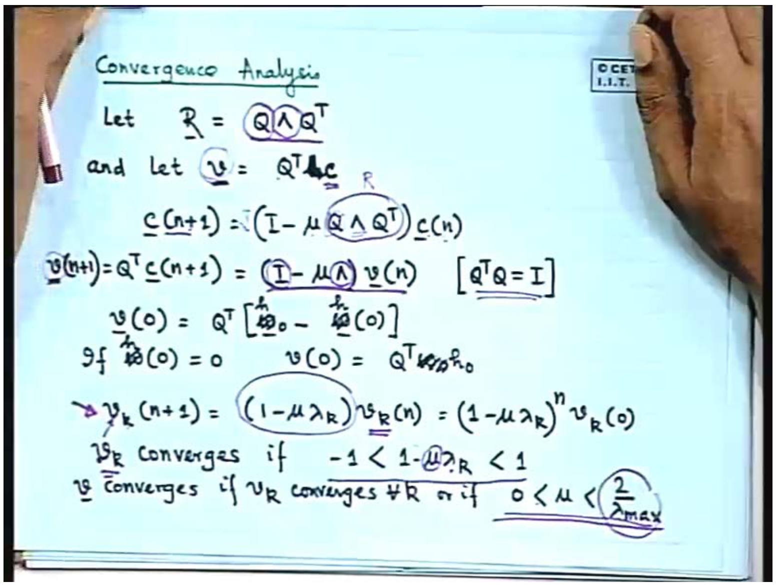
$$c(n+1) = [I - \mu R]\,c(n)$$

$h_0 = R^{-1}d$

$Rc(n) = Rh(n) - d$

Say say c n plus, see the first component of c n plus 1. It will depend on three four components of c n, because this matrix is non-diagonal but now by by by making it on this basis we have reduced it to a diagonal form.

(Refer Slide Time: 27:05)



Convergence Analysis

Let $\underset{=}{R} = Q \Lambda Q^T$

and let $\underline{v} = Q^T \underline{c}$

$\underline{c}(n+1) = (I - \mu Q \Lambda Q^T) \underline{c}(n)$

$\underline{v}(n+1) = Q^T \underline{c}(n+1) = (I - \mu \Lambda) \underline{v}(n) \quad [Q^T Q = I]$

$\underline{v}(0) = Q^T [\hat{\underline{w}}_0 - \hat{\underline{w}}(0)]$

If $\hat{w}(0) = 0 \qquad v(0) = Q^T h_0$

$v_k(n+1) = (1 - \mu \lambda_k) v_k(n) = (1 - \mu \lambda_k)^n v_k(0)$

v_k converges if $-1 < 1 - \mu \lambda_k < 1$

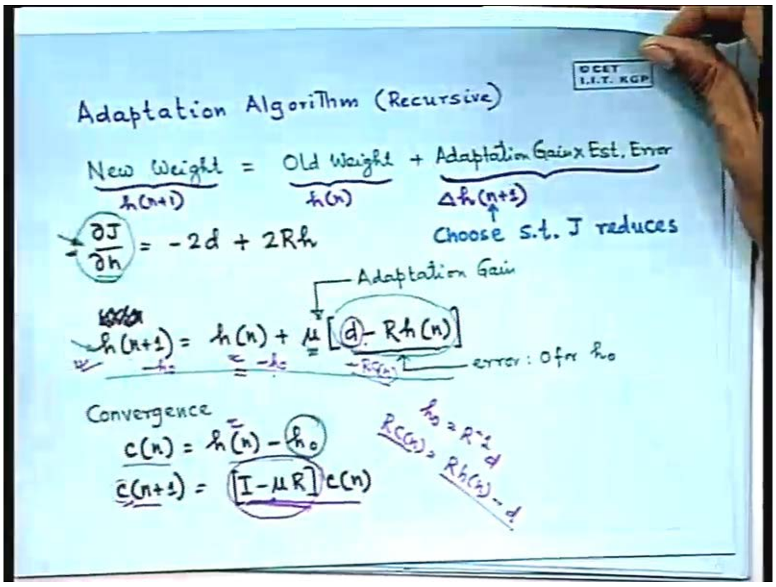
$\underline{v}$ converges if v_k converges $\forall k$ or if $0 < \mu < \frac{2}{\lambda_{max}}$

And now I can say say that, each one of them are basically diagonal. So then we we have a set of scalar equations for for each component of v, we have a set of scalar equation which involves only that component of v nothing else.

So that k is now restricted, which means that; we can simply say that, if if v k n plus 1 is equal to 1 minus mu lambda k of v k n, then then when does v k n go to? When does this go to zero? When this factor is less than one, either… that is the magnitude of that factor must be less than one, then and then every time you are going to multiply by another less than one and then it will finally go to zero, right. So so v k converges if this is the condition. And if the whole of v has to converge, then each v k must converge. Now if each v k must converge then watch how should you choose mu? You see you have chosen mu, mu is a single quantity it does not depend on k, it is the same for each k.

So therefore you should you should choose this mu, such that you should choose the mu corresponding to the maximum Eigen value of lambda. If it satisfies if its satisfied is it for the maximum Eigen value, it will obviously satisfy it for all other Eigen values, right. So then this is the condition that, mu must lie between 0 and 2 by lambda maximum. So if this condition is satisfied, then then then this iteration scheme given enough time, v converge through the optimal weight, right.

(Refer slide Time: 28:43)



©CET I.I.T. KGP

Adaptation Algorithm (Recursive)

$$\underbrace{\text{New Weight}}_{h(n+1)} = \underbrace{\text{Old Weight}}_{h(n)} + \underbrace{\text{Adaptation Gain} \times \text{Est. Error}}_{\Delta h(n+1)}$$

Choose s.t. J reduces

$$\boxed{\frac{\partial J}{\partial h}} = -2d + 2Rh$$

Adaptation Gain

$$h(n+1) = h(n) + \mu\left[d - Rh(n)\right]$$

error : 0 for h_o

Convergence

$$c(n) = h(n) - h_o$$

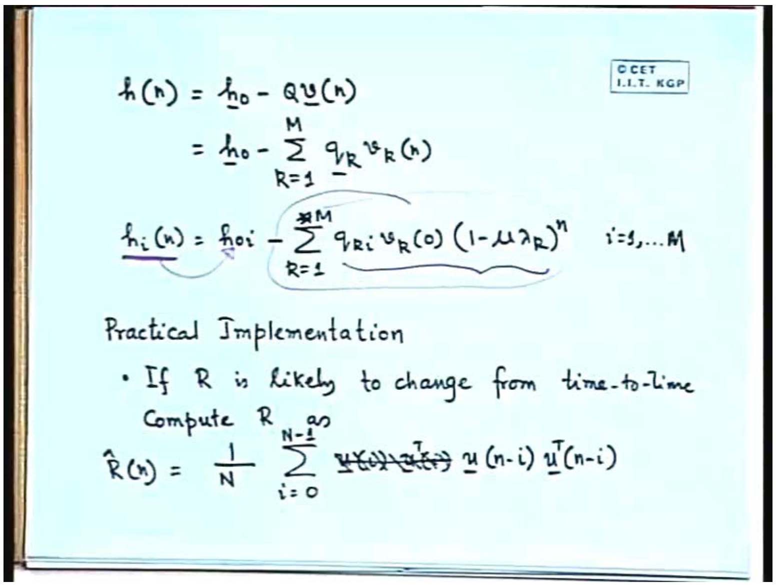
$$c(n+1) = \boxed{[I - \mu R]}\, c(n)$$

$h_o = R^{-1}d$

$Rc(n) = Rh(n) - d$

So this is this is our result. So that is how you should choose the adaptation gain. If you choose the adaptation gain too much, then the process will become unstable, right. Just like control, even in control if you if you increase your controller gain too much, the process becomes unstable. So this is our and now now now we can you know calculate all sorts of things.

(Refer Slide Time: 29:21)



©CET I.I.T. KGP

$$h(n) = h_0 - Qv(n)$$

$$= h_0 - \sum_{R=1}^{M} q_R v_R(n)$$

$$h_i(n) = h_{oi} - \sum_{R=1}^{M} q_{Ri} v_R(0) (1 - \mu\lambda_R)^n \qquad i = 1, \ldots M$$

Practical Implementation

- If R is likely to change from time-to-time Compute R as

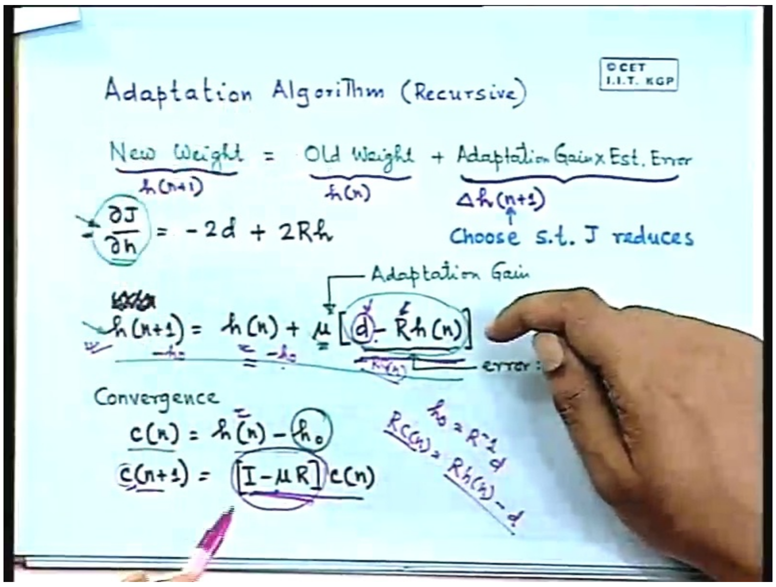
$$\hat{R}(n) = \frac{1}{N} \sum_{i=0}^{N-1} u(n-i)\, u^T(n-i)$$

So hi, that is the ith component of my weight will now converge like this, now we can write everything because because they are now all scalar equations. So you see that as n will increase, this term this whole term will actually go to zero; which means that, this whole sum will eventually go to zero and this will tend to this.

So so so each component of the weight will actually; tend to the corresponding optimal wiener filtering weight. So this is so this algorithm is okay, it at least does the job and hopefully even at given enough time, it will do the job but there are several catches; for example, how do you know R, that is the one of the one of the basic catches, that how do you know R?

Actually, what you what you do is, you you also try to try to continuously and then I mean R may change as you have said. So you have to also continuously update R, it is continuously obtain the latest estimate of R, because you have to if you want to update, then you need R, this is your this is your update.

(Refer Slide Time: 30:31)



Adaptation Algorithm (Recursive)

$$\underbrace{\text{New Weight}}_{h(n+1)} = \underbrace{\text{Old Weight}}_{h(n)} + \underbrace{\text{Adaptation Gain} \times \text{Est. Error}}_{\Delta h(n+1)}$$

Choose s.t. J reduces

$$\frac{\partial J}{\partial h} = -2d + 2Rh$$

Adaptation Gain

$$h(n+1) = h(n) + \mu \left[d - \hat{R} h(n) \right]$$

error

Convergence

$$c(n) = h(n) - h_0$$

$$c(n+1) = \boxed{I - \mu R}\, c(n)$$

$$h_0 = R^{-1} d$$

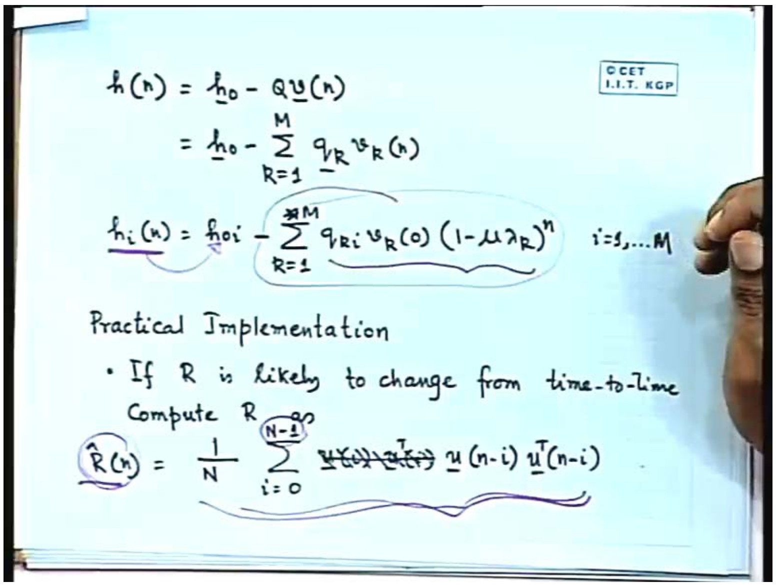
$$Rc(n) = Rh(n) - d$$

So you need d and you need R. Now you are asking that, that if I need, if I get have d and if I have R, why do not I compute the the optimal wiener filtering weight, why not? That is usually because of two reasons, see when we study control, actually everything depends on you know computation speed. If you if you ask a chemical engineer, he has he has one computation is is is is not in his mind at all, because his sampling time in his processes his sampling times will be in minutes. He does not care what amount of time of needed. If you ask a a an electrical engineer, who wants to drive a motor, he will be somewhat concerned but not too much because his sampling times are in milliseconds.

If you ask a communication engineer, he is extremely concerned about computing time. For example, I think he I mean I was speaking with your supervisor, he all his Ph.D. work will be on how to compute faster, all these algorithms. So in signal processing for for for for communication, computation is a big issue because you are sampling at gigahertz frequency, right. Megabytes per second, gigabytes per second kind of things, know telephoning. So there so so you you can imagine that, inverting a big matrices, every time is a big problem, you cannot do that computation every time.

So I mean that way the steepest descent algorithm is I mean the the the computation is very simple. Just one multiplication and and one subtraction, I mean you do not have to solve equations which you have to do, if you want to solve the optimal wiener filtering solution at every step, right. So and later on we will see in fact the the next algorithm, that we will see we will see that, how this computation itself is. Now you still have to compute R; remember that the computation is not only this, computation is this provided you know R and d, but if you want to simultaneously update R and d, then how do you update R and d?

(Refer Slide Time: 32:48)



© CET I.I.T. KGP

$$\underline{h}(n) = \underline{h}_0 - Q\underline{v}(n)$$

$$= \underline{h}_0 - \sum_{R=1}^{M} \underline{q}_R v_R(n)$$

$$\underline{h_i(n)} = h_{0i} - \sum_{R=1}^{M} \underbrace{q_{Ri}\, v_R(0)\, (1-\mu\lambda_R)^n} \qquad i=1,\dots M$$

Practical Implementation

- If R is likely to change from time-to-time Compute R as

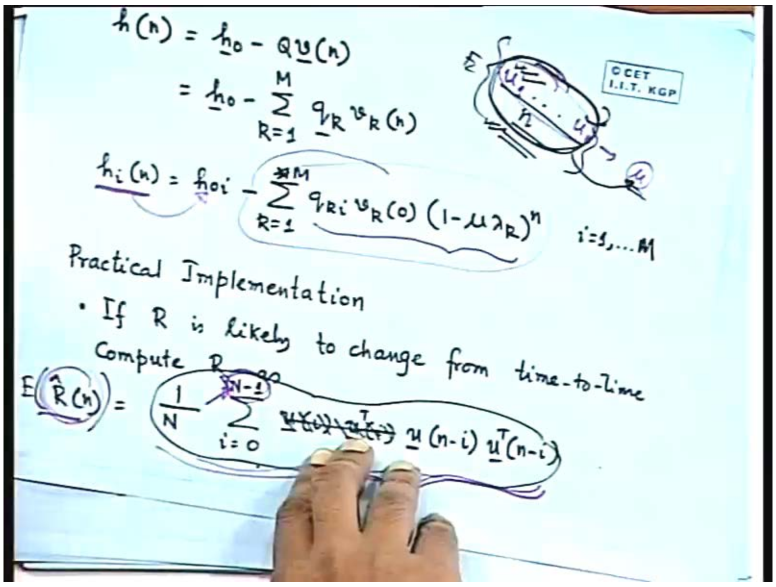
$$\hat{R}(n) = \underbrace{\frac{1}{N} \sum_{i=0}^{N-1} \underline{u}(n-i)\, \underline{u}^T(n-i)}$$

I mean the standard I mean absolutely simplistic way of being R and d will be this, calculate u transpose u, for the last n samples. And see where it where it is just simply take an average. That is that will give you R and d, I mean similarly they will gave you R and you can you can similarly do u n minus I, y n minus i to get d. So this is the very computation intensive thing, if you want to do it every time. Every time if you able to take n number of samples and multiplied to matrices and then add them, that is not a very simple task I mean it it involves computation.

So so so we will still have to see that how we can simplify this, if we actually what they implement. And actually it is this motivation that gives rise to the next algorithm which is the LMS algorithm. So in the LMS algorithm; what are what we are trying to do is we we we are still doing gradient gradient algorithm, only thing is that we want to simplify it. It is just based on computation, upon a just based on computation advantage. That if we we see that, how much of.. see we want to adapt R because we know that from time to time R and d will change. That is that is a that is a basic reason, for choosing to adapt a filter.

So we know that R R and R and d may change. So so we need to adapt R and d but at the same time we are also looking for simple schemes of computation for for R and d. So how do you do that? What is the what is the what is the most simple way of choosing R and d? The the most simple way is to simply reduce this n to one.

(Refer Slide Time: 34:34)



$$\underline{h}(n) = \underline{h}_0 - Q\underline{v}(n)$$

$$= \underline{h}_0 - \sum_{R=1}^{M} \underline{q}_R v_R(n)$$

$$h_i(n) = h_{0i} - \sum_{R=1}^{M} q_{Ri} v_R(0) (1-\mu\lambda_R)^n \qquad i=1,\dots M$$

Practical Implementation

- If R is likely to change from time-to-time Compute R

$$E\left(\hat{R}(n)\right) = \frac{1}{N} \sum_{i=0}^{N-1} \underline{u}(n-i)\, \underline{u}^T(n-i)$$

That is do not take averages, average means every time so many computations will come; just take one, see without taking one you cannot compute R. So that the best that you can do, I mean I mean that is the that that is the least that you can do. So at least just take for that sample and then assume that that is the R, obviously it will not be the R, it will be one outcome of an of an of a stochastic experiment.

After all u is a u is u is the outcome of one experiment. So if you are why actually you should take n sample average, if you want to compute R actually you should take n sample average; you are not doing that, you are assuming ergodicity and you are doing a time average and thinking that if you take in long enough then the time average will be very close to the n sample average, because you cannot do n sample average in real time anyway. You can, people are as some computers are beginning to become faster and faster, people are now thinking of doing real time, you know Monte Carlo kind of algorithms where they they will calculate n sample averages in real time; those algorithms are now being proposed because the computation platforms are becoming very very fast, compare to the application demand times.

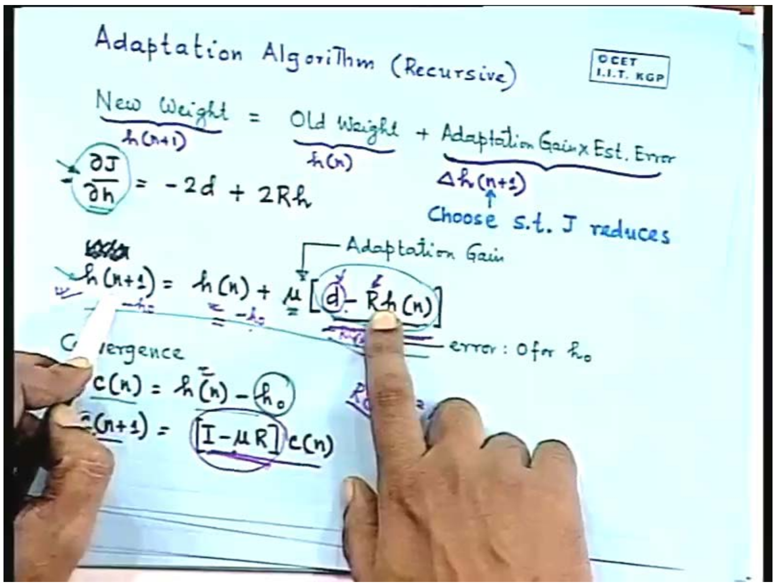
So in some applications such algorithms are also coming, but definitely not in communication. Communication itself is so fast, you are trying to no question of doing n sample average; so you are doing time average, now time average if you now we have seen that, if we take a random variable and if we try try to suppose we suppose you tell a mean, right. We take a random variable, okay several random variables. So if we if we take, this comes from a distribution which has the mean; we do not know what is the mean, okay. All we know, now still we want to still we want to estimate the mean; this happens in so many types. So what is the we, what is that we do? We simply take n of them and then check average and we assume that, that is the mean. Now that is not the mean, that is that is another random variable which has been computed out of these, so that itself has a mean.

This estimate, this mean estimate itself is a is a is a random variable, which has a mean. And that mean converges to mu, true. That is if you take expectation of this, you will also get mu but so the what is the big deal? I mean from from from one random variable, we have got another random variable, is it any better? It is better, because the variance of this random variable is going to be much smaller than, the variance of these.

So now every time you do it, you are going to get something which is very close to mu because the because the variance of this is low. That is the advantage that is why you do averaging. So here also you are doing the same thing, if you do averaging then this quantity will still may not give you R, but because you have done an n point average; it will give you something which is close to R, because the variance is decreasing.

Now if you know what to reduce n, that is reduce the number of turns with which you want to average, then what will happen? Mean will still remain the same, there is an expectation of this will still be R but the variance of this estimate will now increase as.. as you reduce n, you have to pay you have to pay a price somewhere. So that is why, the that is the price precisely which you pay, if you want to if you compute this thing based on only one point; then you are estimated variance will increase, and if you and from from from from sample to sample even if R remains constant from sample to sample, this R n will widely vary, right? And and and this R n, this R n drives your estimate because, that is there in your algorithm.

(Refer Slide Time: 38:44)



Adaptation Algorithm (Recursive)

OCET I.I.T. KGP

$$\underbrace{\text{New Weight}}_{h(n+1)} = \underbrace{\text{Old Weight}}_{h(n)} + \underbrace{\text{Adaptation Gain} \times \text{Est. Error}}_{\Delta h(n+1)}$$

Choose s.t. J reduces

$$\frac{\partial J}{\partial h} = -2d + 2Rh$$

Adaptation Gain

$$h(n+1) = h(n) + \mu\left[d - Rh(n)\right]$$

error : 0 for h_o

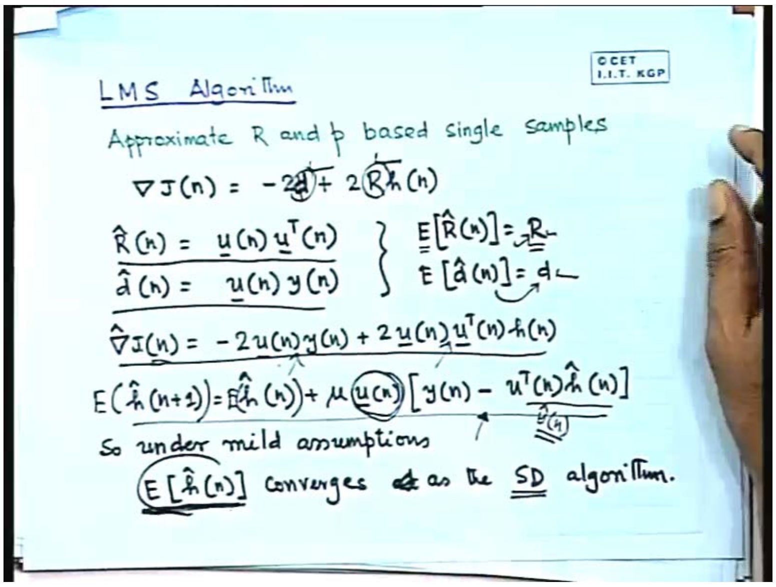
Convergence

$$c(n) = h(n) - h_o$$

$$c(n+1) = \left[I - \mu R\right] c(n)$$

So so so if you have flickering in this R, you are going to have flickering in this. That is the price you people pay in the LMS algorithm, for computational simplicity, right.

(Refer Slide Time: 39:02)



OCET I.I.T. KGP

LMS Algorithm

Approximate R and p based single samples

$$\nabla J(n) = -2\underline{d} + 2\underline{R}\,h(n)$$

$$\left.\begin{aligned}\hat{R}(n) &= \underline{u}(n)\,\underline{u}^T(n)\\ \hat{d}(n) &= \underline{u}(n)\,y(n)\end{aligned}\right\}\quad \begin{aligned}E[\hat{R}(n)] &= \underline{R}\\ E[\hat{d}(n)] &= d\end{aligned}$$

$$\hat{\nabla} J(n) = -2u(n)y(n) + 2\underline{u}(n)\underline{u}^T(n)h(n)$$

$$E(\hat{h}(n+1)) = E(\hat{h}(n)) + \mu\,\underline{u}(n)\left[y(n) - \underbrace{u^T(n)\hat{h}(n)}_{\hat{y}(n)}\right]$$

So under mild assumptions

$E[\hat{h}(n)]$ converges as the SD algorithm.

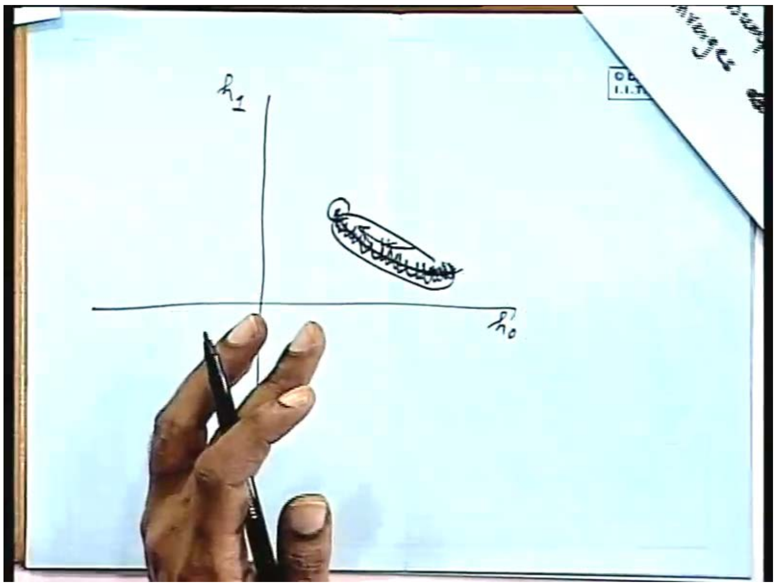
So so what happens is what you do, what you do in even fully knowing this, you want to do it sometimes you know, so there what we say is that, we we estimate R like this. You see, we have I am not take an average, simply one term only u and u transpose n. And then d n is this, simply u n y n again one turn. Turns out that, obviously expectation of R hat n is R, because if you take expectation here, expectation here they will converge, this is will be R. So so they so the mean value remain as R n d, that is true.

So even if they will flicker, their mean will be the R and d which you should have, I mean which we should have used, if you wanted to get a get an get an get an ideal steepest descent; but through a well you are you are sacrificing that. So so now you are performance criterion is is is this, is simple. Just if you that is in place of this, you put this and in place of this you put this. These are your estimates that you are substituting and then you simply. So now this is your rule, if you just put this, the plus mu minus nu, this J n, if you put you will get this one.

So so now you see, what is the what what what are you doing? Look at this, that you are updating, what is your, which is the error that you you are updating on a based on the current value of error, just on the current value of error. So all your estimates, so you so your update is in in this direction and it is based on the current value of the error; because this is what, this is this is nothing but y hat n all right. And obviously now, if you what is the what are the what are the goodness of this?

At least if you take expectation on both sides, this will be expectation and here if you take some mild mild you know conditionsl; if you assume that, things are uncorrelated you can put the expectation inside, and all some some some mild condition it turns out, that this that is these will then converge to their their their corresponding counter parts. And these estimate that is the LMS estimate, expected value of LMS estimate will converge to the corresponding steepest descent algorithm.

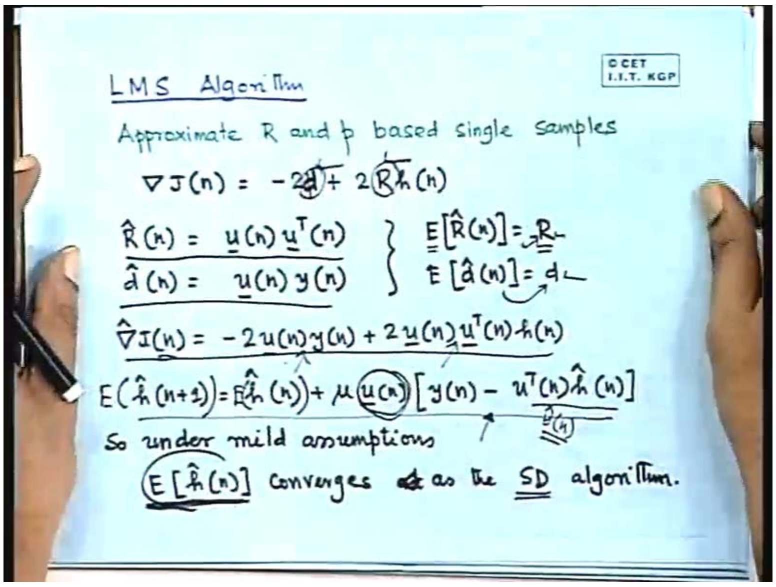
(Refer Slide Time: 41:31)



That is I mean I mean actually speaking, practically if you find that, these are the optimal weights and you have and you have stated, your your say say this is the h zero h ones space suppose; there are only two tow coefficient, that we are that we want to add up. So may be that the that the that the LMS algorithm we will go in this path, then the rather the steepest descent algorithm, if we use the true value of R and d, while if you do an LMS algorithm, it will go like this.

So it will so in in the expectation, it will still follow the steepest descent path but on its journey it will oscillate, because the variance is high, right. But still this is not all that bad, because still it is roughly going to follow these basic steepest descent algorithm which we have seen that, if we define our adaptation weights carefully, which will come fairly close to the optimal weight.

(Refer Slide Time: 42:31)



LMS Algorithm

© CET I.I.T. KGP

Approximate R and p based single samples

$$\nabla J(n) = -2\boxed{d} + 2\boxed{R}h(n)$$

$$\left.\begin{aligned}\hat{R}(n) &= \underline{u}(n)\underline{u}^T(n)\\ \hat{d}(n) &= \underline{u}(n)y(n)\end{aligned}\right\} \quad \begin{aligned}E[\hat{R}(n)] &= R\\ E[\hat{d}(n)] &= d\end{aligned}$$

$$\hat{\nabla}J(n) = -2u(n)y(n) + 2\underline{u}(n)\underline{u}^T(n)h(n)$$

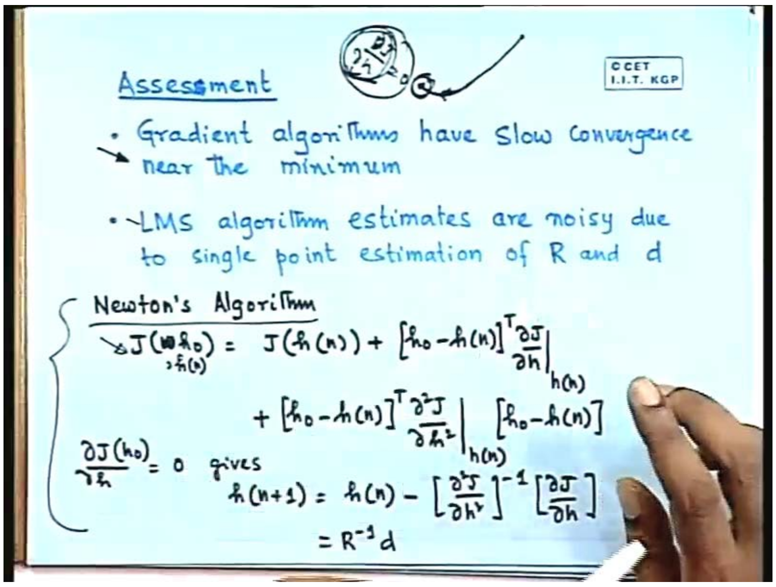
$$E(\hat{h}(n+1)) = E(\hat{h}(n)) + \mu\,\boxed{u(n)}\,[y(n) - \underbrace{u^T(n)\hat{h}(n)}_{\hat{y}(n)}]$$

So under mild assumptions

$\boxed{E[\hat{h}(n)]}$ converges as the SD algorithm.

So at least we we can expect some performance but, there is a basic problem of all gradient algorithms; all gradient algorithms including numerical optimisation, I mean numerical solution algorithms, have this problem, That the gradient algorithms have very slow convergence, near the minimum.

(Refer Slide Time: 42:40)



Assessment

- Gradient algorithms have slow convergence near the minimum
- LMS algorithm estimates are noisy due to single point estimation of R and d

Newton's Algorithm

$$J(h_0) = J(h(n)) + [h_0 - h(n)]^T \frac{\partial J}{\partial h}\bigg|_{h(n)} + [h_0 - h(n)]^T \frac{\partial^2 J}{\partial h^2}\bigg|_{h(n)} [h_0 - h(n)]$$

$\frac{\partial J(h_0)}{\partial h} = 0$ gives

$$h(n+1) = h(n) - \left[\frac{\partial^2 J}{\partial h^2}\right]^{-1}\left[\frac{\partial J}{\partial h}\right] = R^{-1} d$$

So as they so they will suppose, you are here and here is the minimum; initially they will come very close fast, but then towards the end they will become slow, because the magnitude of the gradient itself will becomes smaller. Because in the optimum delta J, that is delta J, delta h is equal to zero, that is how it is a minimum. Now you are now you are updating are also based on this delta J, delta h. So as it comes close to the minimum, this amplitude will becomes smaller. So so every time you have as you are approaching the minimum, you are you are becoming slower, that is the biggest problem of all gradient algorithms including this steepest descent and LMS.

So that is their problem and if you want to there are there are other algorithms which will which will accelerate convergence, let us not look at these; I mean I just sort of wrote it for completeness sake that, there are there are other methods of a you know accelerating convergence, that is by using Newton’s algorithms. That is you do not assume, you do not take only first order terms. Gradient algorithm assumes that, the that the that the performance surface can be approximated as planes. So at any point, it will assume that it can be approximated by a plane. That plane will go on changing but it will intrinsically assume that, the that the surface is the plane.

Now you can you can improve that and you can assume that, the that the surface is the parabola. Thus you can consider the second order differentials in your Taylor series expansion. If you if you define an define an estimator based of that, then you get what is known as Newton’s algorithm, which has much faster convergence but which has much higher computation.

So everywhere when you cannot eat the cake and generally you cannot eat the cake and have it two. So if you are one one faster convergence, you have to spend more money buy faster computers which will do faster computation. So without paying that money, you cannot get everything generally. So so so these are the two major points; firstly that the gradient algorithms have slow convergence near the minimum, and the second is that the LMS algorithms algorithm estimates at generally noisy due to the this single point estimation of R and d, okay.

So that is where we end, so what did we see today? Just to recapitulate, what is we have seen is that, we have first seen optimal estimates in the wiener filtering sense. Now we are seeing schemes where if we start from an initial guess and without ever solving the the wiener filter normal equation, we are just I mean adapting the estimate, right. And we have we are saying that, this will be useful, if if first of all due because of less computation and the second because of the because this R and d parameters, make make make constantly change.

So it is not possible to you know constantly every time evaluate R and d; and try to solve the optimal equations, that is not that is not easy to do, when you have to do it very fast. So these are so they are so we are finding out, computationally efficient adapting structures where the weight surgery is simply adapted and if you give them little time, then they will come very close to the optimal estimates, even otherwise they will constantly be tracking the optimal estimates. So we have seen two algorithms steepest descent and an LMS which are basically gradient based and seen some desirable properties.

So so we are ending our discussions, on what is known as you know adaptive filtering and and linear optimal filtering here. After the exams we have the major topic is Kalman filtering which is state estimation, wiener filtering is just in a sense it is it is input output that is based on some measurements, you want to explain other measurements. But Kalman filtering is estimation of state and then we will have to do something on system identification that is. All these time we are talking about signals, sometimes we have to estimate coefficients in systems. So what is the resistance of a motor, as it is changing or what is the inductance like that or what is some friction coefficient? So, how to do that? So that so that is our plan, so that is all here today.