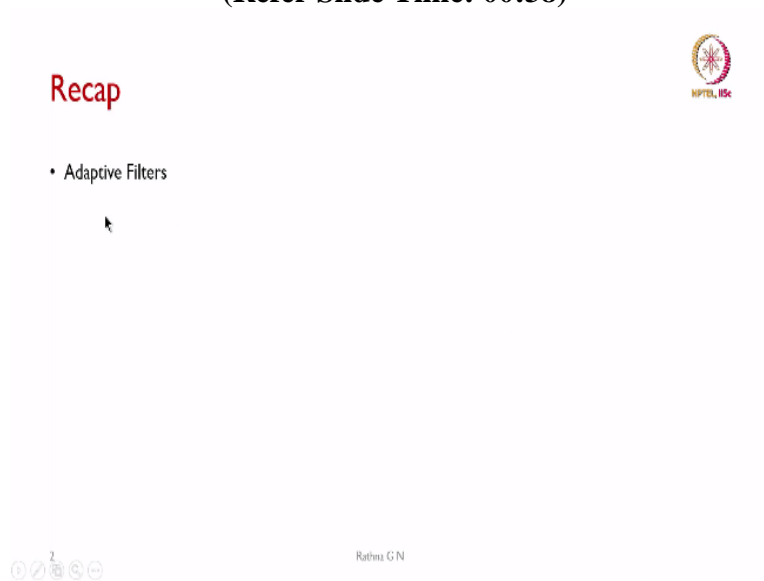


Real - Time Digital Signal Processing
Prof. Rathna G N
Department of Electrical Engineering
Indian Institute of Science - Bengaluru

Lecture - 35
LMS Algorithm

Welcome back to real time digital signal processing course. So last class, we were looking into LMS algorithm. So we will see that how we are going to derive this one in today's class.

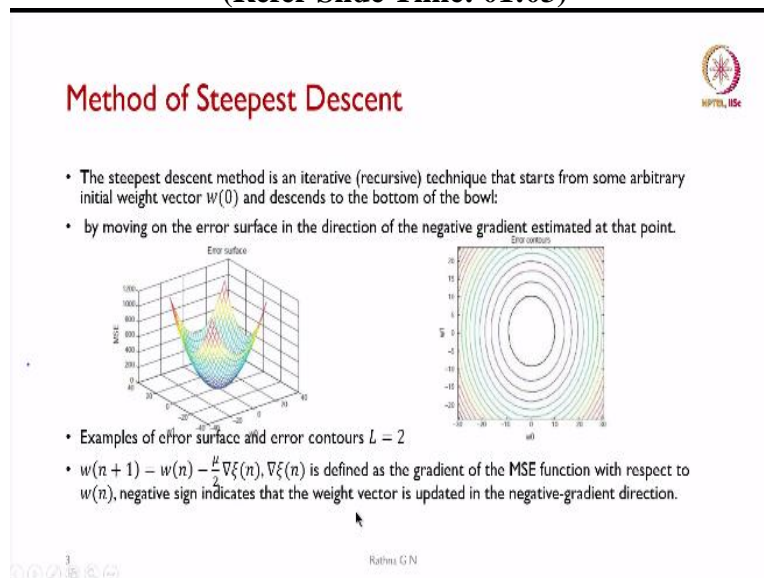
(Refer Slide Time: 00:38)



The slide is titled "Recap" in red. It features a bullet point: "• Adaptive Filters". In the top right corner, there is the NPTEL IISc logo. At the bottom, there are navigation icons and the text "Rathna G N".

So, we have started with the adaptive filters, why do we need it and what are the applications, it is going to be used one of the application, we saw it as hearing the buds and then other things. So, today, we will see that how we can derive adaptive filter that is basically Least Mean Square LMS algorithm.

(Refer Slide Time: 01:03)



The slide is titled "Method of Steepest Descent" in red. It contains the following text and elements:

- The steepest descent method is an iterative (recursive) technique that starts from some arbitrary initial weight vector $w(0)$ and descends to the bottom of the bowl:
- by moving on the error surface in the direction of the negative gradient estimated at that point.

Below the text are two plots:

- A 3D plot titled "Error surface" showing a bowl-shaped surface with axes labeled "MSE" (vertical) and "w" (horizontal).
- A 2D plot titled "Error contours" showing concentric elliptical contours with axes labeled "w" (horizontal) and "MSE" (vertical).

Below the plots, there is more text:


- Examples of error surface and error contours $L = 2$
- $w(n+1) = w(n) - \frac{\mu}{2} \nabla \xi(n)$, $\nabla \xi(n)$ is defined as the gradient of the MSE function with respect to $w(n)$, negative sign indicates that the weight vector is updated in the negative-gradient direction.

At the bottom, there are navigation icons and the text "Rathna G N".

So, we discussed about the method of steepest descent in the last class. So, that is, it is an iterative or recursive process technique that starts from some arbitrary initial weight vector $w(0)$ and it is going to descend to the bottom of the bowl what we said. So, by moving on the error surface in the direction of negative gradient estimated at that point, so, you will be estimating it and then going down in the thing.

So, for $L = 2$ so, this is the error surface what we have got it, and this is the error contours, in concentric circles, what we will be getting at and then how we are going to calculate, we said we will be deriving this shortly. So, that is future weight update $n + 1$ is going to be done with the current weight $-\mu$ is step size basically, as you can see in the error surface, how we will be coming down and then the $\nabla \xi(n)$. So, that is what, what we call it as gradient of the mean square error function with respect to weight function. So, negative sign indicates that the weight vector is updated in the negative gradient direction.

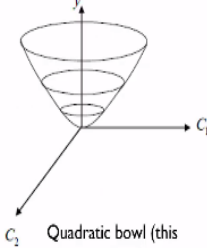
(Refer Slide Time: 02:32)



Example – Steepest Descent

- Given the following function we need to obtain the vector that would give us the absolute minimum.

$$Y(c_1, c_2) = c_1^2 + c_2^2$$
- It is obvious that $c_1 = c_2 = 0$ give us the minimum.



Quadratic bowl (this figure is quadratic error function)

Rathni GN

So, we will take up an example and see how we will be coming with the work on the steepest descent algorithm. So, the function given we need to obtain the vector that would give us the absolute minimum what we are looking at. So, $Y(c_1, c_2) = c_1^2 + c_2^2$, so it is obvious that $c_1 = c_2 = 0$ give us the minimum basically. So, this is the quadratic bowl c_1 is in this direction and c_2 . So, we know that, this is what, what we are expecting. So in the bowl, how will be traversing and coming down what we will see? In practical applications we may not reach 0.

(Refer Slide Time: 03:20)

Example – Steepest Descent (2)



- We start by assuming $(C_1 = 5, C_2 = 7)$
- We select the constant μ . If it is too big, we miss the minimum. If it is too small, it would take us a lot of time to hit the minimum. Select $\mu = 0.1$.
- The gradient vector is

$$\nabla y = \begin{bmatrix} \frac{dy}{dc_1} \\ \frac{dy}{dc_2} \end{bmatrix} = \begin{bmatrix} 2C_1 \\ 2C_2 \end{bmatrix}$$

- So our iterative equation is:

$$\begin{bmatrix} C_1 \\ C_2 \end{bmatrix}_{[n+1]} = \begin{bmatrix} C_1 \\ C_2 \end{bmatrix}_{[n]} - 0.2 * \nabla y = \begin{bmatrix} C_1 \\ C_2 \end{bmatrix}_{[n]} - 0.1 \begin{bmatrix} C_1 \\ C_2 \end{bmatrix}_{[n]} = 0.9 \begin{bmatrix} C_1 \\ C_2 \end{bmatrix}_{[n]}$$



So, as an example so assume that $C_1 = 5$ and $C_2 = 7$ in the beginning and we are going to select the constant μ , if it is too big, we miss the minimum if it is too small, it would take us a

lot of time to hit the minimum. So, in this case, we will select $\mu = 0.1$. The $\nabla y = \begin{bmatrix} \frac{dy}{dc_1} \\ \frac{dy}{dc_2} \end{bmatrix} =$

$$\begin{bmatrix} 2C_1 \\ 2C_2 \end{bmatrix}?$$

So, our iterative equation is what we are going to have it is $\begin{bmatrix} C_1 \\ C_2 \end{bmatrix}_{[n+1]}$ is given as $\begin{bmatrix} C_1 \\ C_2 \end{bmatrix}_{[n]} - 0.2 *$

∇y . So, which is nothing but $\begin{bmatrix} C_1 \\ C_2 \end{bmatrix}_{[n]} - 0.1 \begin{bmatrix} C_1 \\ C_2 \end{bmatrix}_{[n]}$ what we are going to get the thing, so when

we substitute this $1 - 0.1$ will be 0.9 . So, this is 0.2 divided by what you are going to have it is 2 so, that is the reason why what you will be getting as 0.1 .

(Refer Slide Time: 05:03)

Example – Steepest Descent (3)

- Iteration 1: $\begin{bmatrix} C_1 \\ C_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 7 \end{bmatrix}$
- Iteration 2: $\begin{bmatrix} C_1 \\ C_2 \end{bmatrix} = \begin{bmatrix} 4.5 \\ 6.3 \end{bmatrix}$
- Iteration 3: $\begin{bmatrix} C_1 \\ C_2 \end{bmatrix} = \begin{bmatrix} 0.405 \\ 0.567 \end{bmatrix}$
-
- Iteration 60: $\begin{bmatrix} C_1 \\ C_2 \end{bmatrix} = \begin{bmatrix} 0.01 \\ 0.013 \end{bmatrix}$

• As we can see, the vector $[c_1, c_2]$ converges to the value which would yield the function minimum and the speed of this convergence depends on μ .

So, now, what we are going to substitute is iteration first what we have $\begin{bmatrix} C_1 \\ C_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 7 \end{bmatrix}$ is been given, and in the second iteration, so, $\begin{bmatrix} C_1 \\ C_2 \end{bmatrix}$ according to this equation, substitute $\begin{bmatrix} C_1 \\ C_2 \end{bmatrix}$ and then solve the thing $0.9 \begin{bmatrix} C_1 \\ C_2 \end{bmatrix}$, what we are going to have it, so, the next iteration, so, by multiplying $\begin{bmatrix} C_1 \\ C_2 \end{bmatrix}$, so, you will be getting it 0.9 times 5 is going to be 4.5 and this becomes 6.3 and then continue this in the iteration 4.5 and 0.9 so this is what, what we will be getting it and C_2 is this value.

So, at iteration 60 as you can see that the value has come down to 0.01. So, you will be seeing that initial guess what you have taken the thing $\begin{bmatrix} C_1 \\ C_2 \end{bmatrix}$ and then you are traversing down the thing.

So, here it is going to be 0.01 and 0.013. So, as we can see the vector $\begin{bmatrix} C_1 \\ C_2 \end{bmatrix}$ converges to the value, which would yield the function minimum and the speed of this convergence depends on step size. So, in this case, we had taken it as 0.1 is steps times μ basically, that is what, what we have assumed in this case.

(Refer Slide Time: 06:32)

LMS Algorithm



- In many practical applications, the statistics of $d(n)$ and $x(n)$ are unknown. Therefore, the method of the steepest descent cannot be used directly since it assumes the MSE is available to compute the gradient vector. The LMS algorithm developed by Widrow uses the instantaneous squared error, $e^2(n)$, to estimate the MSE as

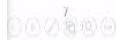
- $\hat{\xi}(n) = e^2(n)$

- Therefore, the gradient estimate is the partial derivative of this cost function with respect to the weight vector as

$$\nabla \hat{\xi}(n) = 2[\nabla e(n)]e(n)$$

- Since $e(n) = d(n) - w^T(n) \times (n)$, $\nabla e(n) = -x(n)$, the gradient estimate becomes

$$\nabla \hat{\xi}(n) = -2x(n)e(n), w(n+1) = w(n) + \mu x(n)e(n)$$



Rafiqul G N

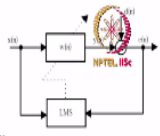
So, now, what is the thing is going to happen? So, how to calculate the mu or how to arrive at the least mean square algorithm, what it is shown here? So, in many practical application, so, statistics of desired signal $d(n)$ and $x(n)$ are going to be unknown that is input signal. So, the method of the steepest descent cannot be used directly, since it assumes the mean square error is available to compute the gradient vector. So, then what happens you will be seeing the LMS algorithm developed by Widrow.

So, you can go to the net and then see he is the one who designed least mean square algorithm uses the instantaneous squared error $e^2(n)$ to estimate the mean square error. So, which is given by $\hat{\xi}(n)$ is you will be putting it as $e^2(n)$. So, the gradient estimate is partial derivative of this cost function with respect to the weight vector, then what happens to our gradient which is going to be given by 2 times gradient of error vector into error vector what well be putting it $e^2(n)$.

So, then since we know that error function is given by $d(n) - y(n)$ substituted with the $w^T(n) \times (n)$, what is substituted, then what we have is $\nabla e(n)$ is given by when you take the gradient of it, which results in $-x(n)$. So, that gradient estimate becomes as it is given by $-2x(n)e(n)$, $w(n+1)$ is going to be $w(n) + \mu x(n)e(n)$. So, from where you have substituted that gradient thing with respect to this, I think you can derive the thing. So the constant μ , what we will be putting it into update vector here.

(Refer Slide Time: 09:00)

LMS Algorithm - 2



- The LMS algorithm is illustrated in Figure and its steps are summarized as follows:
- Determine the values of L , μ and $w(0)$, where L is the length of the filter, μ is the step size, and $w(0)$ is the initial weight vector at time $n = 0$. It is very important to determine these parameter values properly in order to achieve the best performance of the LMS algorithm.
- Compute the adaptive filter output as

$$y(n) = \mathbf{w}^T(n)x(n) = \sum_{l=0}^{L-1} w_l(n)x(n-l)$$
- Compute the error signal as $e(n) = d(n) - y(n)$
- Update the weight vector using LMS algorithm, which can be expressed using the following scalar form:

$$w_l(n+1) = w_l(n) + \mu x(n-l)e(n), \quad \text{for } l = 0, 1, \dots, L-1$$
- $2L$ additions and $2L + 1$ multiplications

Rafiq G N

So, then how we are going to calculate LMS algorithm. So, the diagram on the right side it shows that it has an input and then we have the weight vector here. And then $y(n)$ is output, and this is the desired signal, the difference between the 2 will give us the error, which is fed to LMS algorithm, which will modify the weights based on input $x(n)$. So, this is what the figure shows, and will write down the steps which are going to be followed in the algorithm calculation or weight calculation and then the output calculation and error.

So, we are going to determine the values of L is the order of the filter, μ is the weight vector and $w(0)$ is the initial values for weight vector. So, that is what it gives the thing this is the step size and $w(0)$ is the initial weight vector at time $n = 0$. So, it is very important to determine these parameters values properly in order to achieve the best performance of the LMS algorithm. So, we will compute the adaptive filter output as $y(n) = \mathbf{w}^T(n)x(n)$. So, we know that it is a $\mathbf{w}^T(n) * x(n)$.


So, which is nothing but it is $l = 0, 1, \dots, L - 1$ $w_l(n) * x(n - l)$. And we will be computing the error signal as $e(n)$ is given by in the second step $d(n) - y(n)$. So, we have calculated $y(n)$ and weight vector initially we have assumed as 0 and then we will start with it and then whatever the desired signal what we have given minus the output will be the error initially as we know that $y(n)$ may be 0. So, error will be very high, because we have assumed the weight to be 0 and then $x(n)$.

So, error will be high and then you will be seeing once it starts coming down it will be adapting to weights are adjusted. This is how we will be calculating $w_l(n + 1)$ is going to be given by $w_l(n) + \mu x(n - l)e(n)$. So, where l is the length of the filter which is going to vary from 0 to $L - 1$. So, hence you will be seeing that number of computations what it is going to have in calculating the LMS algorithm is $2L$ additions and $2L + 1$ multiplications what we are supposed to 2 to get the output. So, that is what the computation complexity of the LMS algorithm.

(Refer Slide Time: 12:19)


LMS-Convergence Graph

- Example for the Unknown Channel of 2nd order:
- This graph illustrates the LMS algorithm. Start from guessing the TAP weights. Start going in opposite the gradient vector, to calculate the next taps, and so on, until we MMSE is generated, meaning the MSE is 0 or a very close value to it. (In practice we can not get exactly error of 0 because the noise is a random process, we could only decrease the error below a desired minimum)



9
⏪
⏩
⏴
⏵

Ratna GN



So, coming to the thing how graph is going to converge? So, we will see the thing unknown channel of second order what we have chosen the thing and then this graph is going to illustrate that is what you have the initial guess here and then this is c_1 and then c_2 what we have selected initially as we said, So, then what happens, so, it will be traversing depending on μ . will be traversing in this way in the concentric circle, and try to we are supposed to get to 0. So, as we know that it is in practical situations, it is not possible to achieve this, so, we could only decrease the error below a desired minimum, so that we will be able to work on it.

(Refer Slide Time: 13:15)

Modified LMS Algorithms

- 3 Types to improve computation
 - Sign-Error LMS algorithm. $w(n+1) = w(n) + \mu x(n) \text{sgn}[e(n)]$
where
 - Sign-Data LMS algorithm.

$$\text{sgn}[e(n)] \equiv \begin{cases} 1, & e(n) \geq 0 \\ -1, & e(n) < 0 \end{cases}$$

$$w(n+1) = w(n) + \mu e(n) \text{sgn}[x(n)]$$
 - Sign-Sign LMS algorithm

This algorithm requires no multiplication and is designed for VLSI or ASIC implementation to save multiplications.
It is used in Adaptive Differential Pulse Code Modulation (ADPCM) for speech compression.

Rathna G N

So, there are different modified LMS algorithms to reduce computation time, what are the things? There are 3 types to improve the computation, 1 is the sign error LMS algorithm here as you can see, $w(n+1)$ is given as $w(n) + \mu x(n)$ and we will be taking the sign of the error n so, sign data LMS algorithm the other one sign of $e(n)$ is given as 1 or -1. So, if error is greater than or equal to 0 we make it 1 if it is less than 0 it is going to be -1. So, that it is either a negative of it or positive.

So, that we are avoiding the multiplication by error of n in this case. So, the other one is sign-sign LMS algorithm here it is based on the sign error. The other one is on the sign on the data what you are going to have it then in this case what happens sign of $x(n)$ so, that you are reducing your computation. The other one is sign-sign LMS algorithm. So, this algorithm requires no multiplication and a design for mostly VLSI or ASIC implementation to save multiplications

So, in Adaptive Differential Pulse Code Modulation that is ADPCM used for this algorithm for speech compression so, I Think we will be taking the speech coding a little later. So we will see why we need the compression there.

(Refer Slide Time: 15:08)

LMS for Complex Signals



Some practical applications dealing with complex signals and the frequency-domain adaptive filtering require complex operations to maintain their phase relationships.

The complex adaptive filter uses the complex input vector $x(n)$ and complex coefficient vector $w(n)$ expressed as

$$x(n) = x_r(n) + jx_i(n)$$

$$w(n) = w_r(n) + jw_i(n),$$

The complex output signal $y(n)$ is computed as

$$y(n) = w^T(n)x(n),$$

Where all multiplications and additions are complex operations. The complex LMS algorithm adapts the real and imaginary parts of $w(n)$ simultaneously as

$$w(n+1) = w(n) + \mu e(n)x'(n)$$

Adaptive channel equalizers.

$$x'(n) = x_r(n) - jx_i(n)$$



Ratna G N

So, how it is going to work for a complex signals? That is LMS algorithm we have seen the DFT and other things for the complex signals. Here also we have to see that how it is going to work from least mean square applications dealing with complex signals. The frequency domain adaptive filtering require complex operations to maintain their phase relationships in this case, and then the complex adaptive filter uses that is complex vector $x(n)$ and complex coefficients $w(n)$.

So, then $x(n)$ is a $x_r(n) + jx_i(n)$. And w of n is also represented both in real and then imaginary parts in this way, then we know that complex output signal $y(n)$ is computed as $y(n) = w^T(n)x(n)$ what we have it. So, where all multiplications additions are going to be complex operations. So, the complex LMS algorithm adapts the real and imaginary parts of $w(n)$ simultaneously as in this fashion $w(n) + \mu e(n)x^*(n)$. So, adaptive channel equalizers use $x^*(n) = x_r(n) - jx_i(n)$. So, we will see these applications little later.

(Refer Slide Time: 16:51)

Performance Analysis



Stability, convergence rate, excess mean-square error and finite-wordlength effects.

STABILITY CONSTRAINT

$0 < \mu < 2/\lambda_{max}$, λ_{max} is the largest eigenvalue of the autocorrelation matrix R

Where λ_i are the eigenvalues of matrix R , and

$0 < \mu < 2/LP_x$, $P_x \equiv r_{xx}(0) = E[x^2(n)]$ is the power of $x(n)$.

This equation provides two important principles for determining the value of μ

1. The upper bound of the step size μ is inversely proportional to filter length L , thus a smaller μ must be used for a higher order filter, and vice versa.
2. Since the step size is inversely proportional to the input signal power, a larger μ can be used for a low-power signal, and vice versa. A more effective technique is to normalise the step size μ with respect to P , such that the convergence rate of the algorithm is independent of μ

So, we have to see the performance analysis of LMS algorithm the as we know that IIR filter we consider the stability constraints. So, we have to see, first one is the stability. Next is how we are going to have the convergence rate? And then we will be seeing that excess mean square error and how it is going to have the finite-wordlength effects on algorithm just like any other linear filter, we have to see in that adaptive filter also. So, how these parameters are going to affect us?

So, the first one is the stability constraint. So, that is $0 < \mu < 2/\lambda_{max}$, lambda max is the largest Eigen value of the autocorrelation matrix R . So, as we will be seeing in the lab that we selected with $\mu = 1$, how are output was getting affected? So, this is the wave otherwise, arbitrarily we can choose the thing and then what is the thing is going to happen?

So, we have looked in the lab. So, here what is should be the μ value, which should be $< 2/LP_x$, lambda max is Eigen value for autocorrelation matrix R . So, we know that computing autocorrelation matrix and finding out the Eigen value is compute intensive. And then how we are going to take the thing that is lambda L are the Eigen values of matrix R in that the maximum value of what we will be taking it and then divide 2 by that value.

So, the other way of doing selecting μ is from the stability point of view. So, we can have a $\mu < 2/LP_x$, L is the length of the filter and then P_x is we know that it is the autocorrelation function of the first this think input signal which is nothing but $E[x^2(n)]$ is the power of $x(n)$. So, this equation provides 2 important principles for determining the value of μ from these 2 constraints, what is it?

The upper bound of the step size μ is inversely proportional to the filter length here, thus a smaller μ must be used for a higher order filter and vice versa. Since the step size is inversely proportional to the input signal power, a larger μ can be used for low power signal and vice versa. The other way is more effective technique is to normalise the step size μ with respect to power P such that the convergence rate of the algorithm is independent of μ that is what one has to look at it.

(Refer Slide Time: 20:13)

Convergence Speed

Each adaptive mode has its own time constant for convergence, which is determined by the step size μ and the eigenvalue associated λ_i with that mode. Thus, the time needed for convergence is clearly limited by the slowest mode caused by the minimum eigenvalue, and can be approximated as

$$\tau_{mse} \cong \frac{1}{\mu \lambda_{min}}$$

where λ_{min} is the minimum eigenvalue of the matrix R . Because τ_{mse} is inversely proportionate to the step size μ , using a smaller μ will result in a larger τ_{mse} (slower convergence). when τ_{mse} is very large, only a small μ can satisfy the stability constraint

λ_{min} is very small, the time constant can be very large, resulting in very slow convergence. The slowest convergence occurs when using the smallest step size $\mu = 1/\lambda_{max}$

$$\tau_{mse} \leq \frac{\lambda_{max}}{\lambda_{min}}, \quad \tau_{mse} \leq \frac{\lambda_{max}}{\lambda_{min} \cong \frac{\max |X(\omega)|^2}{\min |X(\omega)|^2}}$$

where $X(\omega)$ is the DTFT of $x(n)$. The eigenvalue spread can be efficiently approximated by the spectral dynamic range.

Rafiq G.N

So, now coming with the convergence speed continuing with the thing, so, each adaptive mode has its own time constant for convergence. So, which is going to be determined by the; step size mu as we know and the Eigen value associated with lambda l with that mode. Thus the time needed for convergence is clearly limited by the slowest mode caused by the minimum Eigen value and can be approximated as time to calculate mean square error is approximated as $\frac{1}{\mu \lambda_{min}}$.

So, lambda minimum is the minimum Eigen value of the matrix R, what we are considering this is the maximum time what it is going to take place. If we choose a lambda max, then we know that time to compute this is going to be lesser. So, we say because time tau mean square error is inversely proportional to the step size mu using a smaller mu will result in a larger time basically that is slower convergence.


And when tau max is very large, only a small mu can satisfy the stability constraint. So, these are the things one has to consider when you are selecting your mu. So, if lambda minimum is

very small, that time constant can be very large, resulting in very slow convergence here also. The slowest convergence occurs when using the smallest step size $\mu = 1/\lambda_{max}$. So, you will be seeing that both of them are inversely proportional to mu step size and then λ_{min} .

So, both will be contributing to the slow computation, time for adaptive filter. So, then what is it we say that τ_{mse} is given by less than or equal to $\frac{\lambda_{max}}{\lambda_{min}}$ and which is by substituting λ_{min} we can do that, which is less than or equal to lambda max divided by minimum is approximated as maximum $X(\omega)$ that is the $\frac{max|X(\omega)|^2}{min|X(\omega)|^2}$. So, how is this?

We know that $X(\omega)$ is the DFTF $x(n)$. So, we will be seeing that Eigen value spread can be efficiently approximated by the spectral dynamic range in this case, so we can see how the computation is going to be more?


(Refer Slide Time: 23:10)



Excess Mean-Square Error

$$\xi_{\text{excess}} \approx LP_x \xi_{\text{min}}$$

This approximation shows that the excess MSE is directly proportional to μ .
 Thus the use of larger step size μ results in a faster convergence rate at the cost of degraded steady-state performance by producing more noise.
 Therefore, there is a design trade-off between the excess MSE and the convergence speed when choosing the value of μ .



So, now, the other way of is how we can control mean square error by using excess mean square error. So, which is ξ_{excess} is given us $LP_x \xi_{\text{min}}$. So, we know that, so the approximations shows that the excess mean square is directly proportional to mu and then use of larger steps saves mu is going to result in a faster convergence rate at the cost of degraded steady state performance by producing more noise, this is what, what we saw in the lab?

Or yourself see that by making a μ larger step size only it will be giving you noise. So, therefore, there is a design trade off between the excess mean square error and the convergence speed when choosing the value of μ .

(Refer Slide Time: 24:10)

Normalized LMS Algorithm (NLMS)



$$0 < \mu < \frac{L}{P_x},$$
$$w(n+1) = w(n) + \mu(n)x(n)e(n),$$

where $\mu(n)$ is the time-varying step size normalized by the filter length and signal power as

$$\mu(n) = \frac{\alpha}{L\hat{P}_x(n) + c}$$

where $\hat{P}_x(n)$ is the estimate of the power of $x(n)$ at time n , $0 < \alpha < 2$ is a constant, and c is a very small constant to prevent division by zero or using a very large step size for a very weak signal at time n . Note that $\hat{P}_x(n)$ can be estimated recursively.



Rathna GN


So, now, how we can normalize this mu basically step function, so then later on what it was developed is normalized LMS algorithm. So, which is given by this equation that is $0 < \mu < \frac{L}{P_x}$. So, as it L is the length of this thing filter and P_x is the power what we have to calculate. So, you will be seeing that it will take more time than LMS algorithm, then your update function for $w(n+1)$ is given selecting this $\mu(n)x(n)e(n)$.

So, you will be seeing that where $\mu(n)$ is the time varying step size one has to calculate it is not pre computed as in when you are depending on your input power you will be calculating your step size and then it can be normalized by the filter length and signal power as $\mu(n)$ is given as alpha constant divided by $LP_x(n) + c$, where $P_x(n)$ is the estimate of the power of $x(n)$ at time n .


So, α will be taking the value between 0 and 2 which is a constant and c is very small constant. So, that we are not going to have when this becomes 0 division by 0 is avoided by this constant. So, very small so that this is not going to get affected or using a very large step size for a very weak signal at time n , one has to note that $P_x(n)$ can be estimated recursively.

(Refer Slide Time: 26:13)

NLMS Implementation Consideration




1. Choose $\hat{P}_x(0)$ as the best a priori estimate of the input signal power.
2. A Software constraint may be required to ensure that $\mu(n)$ is bounded if it is very small when the signal is absent.



So, NLMS algorithm will take little time compared to LMS. So, how we are going to choose this $\hat{P}_x(0)$ as the best a priori estimate of input signal power. So, a software constant may be required to ensure that $\mu(n)$ is bounded, if it is very small when the signal is going to be absent, so, that divided by 0 is avoided.

(Refer Slide Time: 26:38)

Software Implementation



```


uen=mu*en;                                     //perform outside loop
For (l=0; l<L, l++)                             //l=0, 1, ..., L-1
{
w[l]=uen*x[l];
}

```

The up-dation can be done once in sampling period, instead of every sample to reduce computation and to take care of the pipelining delays.

The delayed LMS algorithm expressed as

$$w(n+1) = w(n) + \mu e(n-\Delta)x(n-\Delta)$$

$$\Delta = 1$$



So, how we are going to do this one software implementation is shown in with few steps, that is, we will be assigning $\mu e(n)$ is assigned as $\mu e(n)$ for $l = 0, 1$ less than length of the filter. So, you will be continuing it this is the $w(1)$ is calculated with respect to μ into whatever you have calculate $\mu e(n) * x(1)$. So that you need not have to do this multiplication inside every time which is consumed, you know that within the loop if you do the multiplication it will be adding on.

Instead of that you compute and keep it outside and then use it for you will be reducing the multiplication. So, you can see that the updation can be done once in a sampling period what we will be doing it? Instead of every sample to reduce the computation and to take care of the pipelining delays. So, that delayed LMS algorithm which is expressed as $w(n + 1)$ is nothing but $w(n) + \mu e(n - \Delta)x(n - \Delta)$.


So, if you assume $\Delta = 1$ then $e(n - 1)$ that is previous error what you will consider and then even $x(n - 1)$ will be the previous sample what you will be calculating to update weights in the future of it along with the current weight.

(Refer Slide Time: 28:41)

Finite Precision Effects



- Input range is in between -1 and 1 .
- Scaling, finite word effects and arithmetic errors can be considered
- $e(n)$, feedback path for coefficients, scaling is complicated.
- Also, the dynamic range of the filter output is determined by the time-varying filter coefficients, which are unknown at the design stage.
- For adaptive FIR filtering with the LMS algorithm, the scaling of the filter output and coefficients can be achieved by scaling the "desired" signal, $d(n)$. The scaling factor a , where $0 < a < 1$, is used to prevent overflow of the filter coefficients during the coefficient update. Reducing the magnitude of $d(n)$ reduces the gain demand on the filter, thereby reducing the magnitude of the coefficient values. Since a only scales the desired signal, it will not affect the convergence rate, which depends on the magnitude spectrum and power of input signal $x(n)$.



Rathna G N

So, next one is the finite precision effects one has to consider. So, what is this we have seen this effect in case of IIR filter, the same thing we will be using it so, we assumed the range input is going to be in the range between -1 and 1 . So, we will be scaling a signal so, that the input if it is a sine wave the magnitude is going to be between -1 and 1 . So, any even the cos function if you are considered or any speech signal or any audio signal you will be considering the magnitude to be between -1 and 1 .

So, later on we had to do scaling, finite word effects and arithmetic errors can be considered. So, however, we have done in the case of IIR filter, so we will be taking care of the scaling and then finite word effects representing our coefficients even μ has to be approximated after the thing and then whatever additional what we are doing it the errors have to be considered. So we know that $e(n)$ that is feedback for coefficients, scaling is going to be complicated. So, what is it earlier in the FIR filter, we did not have the feedback.

So, you will be seeing in the figure, we have from the error, we are having a feedback path in the thing so, this also has to be considered. Then what happens? Also the dynamic range of the filter output is determined by the time varying filter coefficients which are unknown at the design stage in the FIR filter we have computed coefficients and then we knew that what will be the maximum precision what we are going to get it here what is the thing is going to happen is because we are on the go weight function is getting updated.

So, we say that it is a time varying filter coefficients what we have to calculate? And we do not know at the design stage whether they are going to overflow or underflow one has to consider that. So, we say for adaptive FIR filtering with LMS algorithm, the scaling of the filter output and coefficients can be achieved by scaling that desired signal that is $d(n)$. So, the scaling factor α , which is in between 0 and 1 is used to prevent overflow of the filter coefficients during the coefficient update.

So, reducing the magnitude of $d(n)$ reduces the gain demand on the filter, thereby reducing the magnitude of the coefficient values. Since α only scales the desired signal, so it is not going to affect the convergence rate, which depends on the magnitude spectrum and power of input signal. So that is $x(n)$ so $d(n)$ is not going to come in to spectral calculation or power calculation. So, it is not going to degrade our performance, when we do the scaling of desired signal. So $y(n) - d(n)$ so, which is going to be kept in not to overflow.

(Refer Slide Time: 32:25)

Finite Precision Effects (I)



- The finite-precision LMS algorithm can be described as follows using rounding operations:

$$y(n) = R \left[\sum_{l=0}^{L-1} w_l(n)x(n-l) \right]$$

$$e(n) = R[\alpha d(n) - y(n)]$$

$$w_l(n+1) = R[w_l(n) + \mu x(n-l)e(n)], \quad l = 0, 1, \dots, L-1,$$

where $R[x]$ denotes the fixed-point rounding of the quantity x .

- When updating coefficients, the product $\mu x[n]e[n]$ is a double precision number, which is added to the original stored weight value, $w_l[n]$, and then rounded to obtain the updated value, $w_l[n+1]$.
- Adaptation will stop when this update term is rounded to zero if its value is smaller than the LSB of the hardware. This phenomenon is known as "stalling" or "lockup".
- This problem may be solved by using more bits and/or using larger step size μ , to guarantee convergence of the algorithm. However, using a large step size will increase excess MSE.

So, what will be the thing now continuing with the finite precision LMS algorithm can be how it is going to be described that is using the rounding operations. So, we know that $y(n)$ is equal to that is value is going to be rounded. So, here it is not the autocorrelation matrix what we are representing it with is the rounding. So, the magnitude of whatever the equation we have FIR filter equation, which is going to be rounded.

Then $e(n)$ is going to be resulted as rounded of a times and $d(n)$ that is scaling of desired signal minus $y(n)$. The magnitude of it will be rounding it then what happens to weight function updation function, so that is also going to be rounded. And then you will be having $w_l(n+1) = R[w_l(n) + \mu x(n-l)e(n)]$, $l = 0, 1, \dots, L-1$. So that is what, what it says is $R[x]$ in this case, fixed point rounding of the quantity x .

So when updating coefficients, the product, whatever product we have, it is a double precision number because $a(n)$ is floating point number even μ is also going to be floating point and $x(n)$ is also going to be either fixed point or floating point, still the result of 2 floating point numbers has to be double precision FIR used 32 bit for both of them, then it becomes 64 bit $w_l(n)$ and then rounded to obtain the updated value $w_l(n+1)$.

So, adaptation will stop when this update term is rounded to 0 if its value is smaller than the LSB of the hardware, what we consider this phenomenon is known as stalling or lockup. So, this problem may be solved by using more bits and are using larger step size μ , to guarantee that convergence of the algorithm. However, we know that using a larger step size will increase excess mean square error.

(Refer Slide Time: 35:13)

Leaky LMS algorithm



- The leaky LMS algorithm may be used to reduce numerical errors accumulated in the filter coefficients
- This algorithm prevents coefficient update overflow from finite-precision implementation by providing a compromise between minimizing the MSE and constraining the energy of the adaptive filter. The leaky LMS algorithm can be expressed as
$$w(n+1) = vw(n) + \mu x(n)e(n),$$
- Where v is the leakage factor, $0 < v < 1$: The leaky LMS algorithm not only prevents unconstrained weight overflow, but also limits the power of output signal $y(n)$ in order to avoid nonlinear distortion of transducers (such as loudspeakers) driven by the filter output.
- The excess power of errors caused by the leakage is proportional to $[(1-v)/\mu]^2$
- Therefore, $(1-v)$ should be kept smaller than μ in order to maintain an acceptable level of performance.

So, the other variant of LMS algorithm is the leaky LMS algorithm. So, this is defined that is to reduce numerical errors accumulated in the filter coefficients. So, this algorithm prevents coefficient update overflow from the finite precision implementation by providing a compromise between minimizing the mean square error and constraining the energy of the adaptive filter. So, which is given by expression is given us v times $w(n+1) = vw(n) + \mu x(n)e(n)$.

So, where v is leakage factor 1 is going to consider it is going to be between 0 and 1. So, if it is 1 we know that it is going to be LMS algorithm. So, that leaky LMS algorithm not only prevents unconstrained weight for flow, but also limits the power of output signal y of n in order to avoid nonlinear distortion of the transducer basically, such as loudspeakers driven by the filter output. So, we know that, when we are having a speech or audio the thing the output is going to loudspeakers.

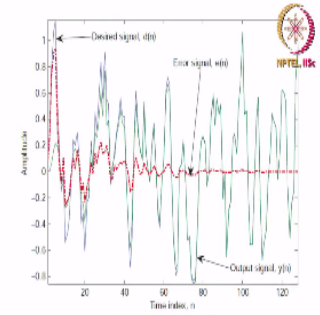
So, this distortion is going to be reduced by using leaky LMS algorithm. The excess power of errors caused by the leakage is proportional to what we call it as $1[(1-v)/\mu]^2$. So, $1-v$ should be kept smaller than μ in order to maintain an acceptable level of performance.

(Refer Slide Time: 37:05)

Adaptive FIR Filter Objects

Object	Algorithm	Description
adaptfilt.lms		Direct-form, (leaky) LMS algorithm
adaptfilt.scl		Direct-form, sign-data LMS algorithm
adaptfilt.scf		Direct-form, sign-error LMS algorithm
adaptfilt.lss		Direct-form, sign-sign LMS algorithm
adaptfilt.nlms		Direct-form, normalized LMS algorithm
adaptfilt.dlms		Direct-form, delayed LMS algorithm
adaptfilt.blms		Direct-form, LMS algorithm

Various LMS type algorithms



```

randn('seed', 12345) % Seed for noise generator
x=randn(1,120) % Reference input signal x(n)
b=[0.1,0.2,0.4,0.2,0.1]; % An FIR filter to be identified
d=filter(b,1,x); % Desired signal d(n)
mu=0.05; % Step size mu
h=adaptfilt.lms(5,mu); % FIR filter with LMS algorithm
[y,e]=filter(h,x,d); % adaptive filtering

```

Rathna GN

This is adaptive FIR filter objects, if you refer to the book here the book is going to be given. So, you will be seeing that adaptive filter dot LMS these are the functions one can use for adaptive filter that is direct form leaky LMS algorithm. So, different methods have been given, one has to keep it in mind that this adaptive filter dot LMS has been removed in the latest version of MATLAB that is 2020 b. So, you have to use it as DSP dot LMS filter.

So and then properly modify the codes and then you are to use it in your assignments or your work when you want to implement these filters. So, the code what it is given that is you will be having the random seed what it is generated, and then you have been given the coefficients of b so, you will be filtering the signal and then you are assigning your mu value as 0.05 and then call this function which has to be modified.

So, we will see how we are going to modify these to the 2020 pre MATLAB and then use this filter to filter the things so, these are the steps. So, when you observe that this is the desired signal $d(n)$ what it is plot it and then you will be seeing that error signal which was very high which is shown in red, which is going to slow down and then you will be what is it almost minimized here. So, this is what your output signal $y(n)$ is going to look like and error signal.

So, initially you will be seeing that output signal has little bit of noise and other things. So, when error is minimized, so it will be following the input signal. So, in the next class we will be seeing adaptive filter applications. So, thank you for listening to this lecture.