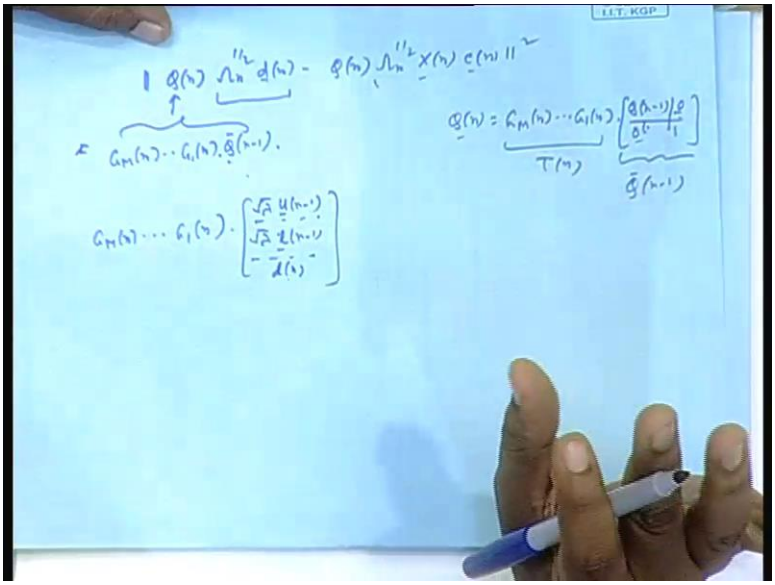
Adaptive Signal Processing
Prof. M. Chakraborty
Department of Electrical & Electronic Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 38
Systolic Implementation (Contd.)

Today, we complete this topic systolic implementation. Actually, it is nothing much, it is more of hardware now.

(Refer Slide Time: 01:07)



What we are doing that time is just quickly Qn, this one, we are minimizing and we said that Qn should be such, that this matrix is triangularied, and you find out the solution for the upper triangular part. Those will give optimal filter coefficients that are standard. Actually, we had this matrix. Qn was, what was our Qn? Qn was sequence of rotations Gmn dot dot dot G 1 n, followed by this matrix, Qn minus 1 0 transpose 0 1, which I called Q bar n minus 1.

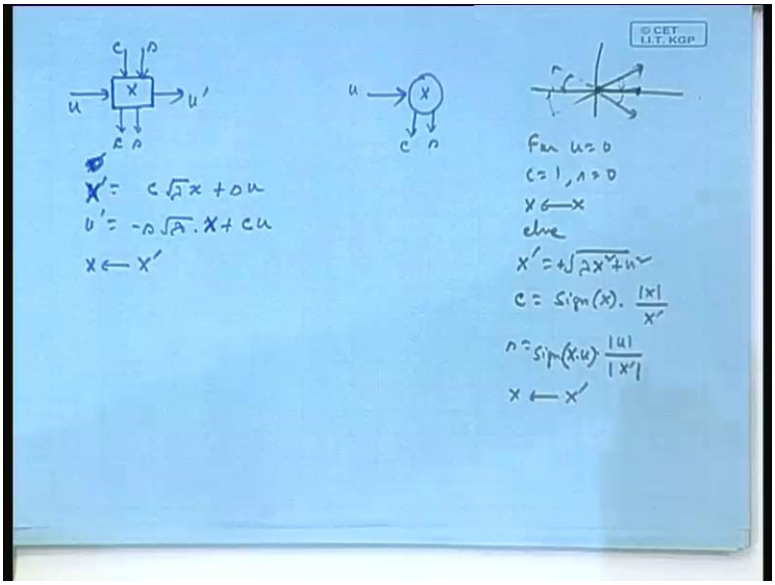
This is Qn, this will work on this part, will give you square root lambda times Rn and then a chunk of 0s, and followed by the last row, will consist of the new entries. This matrix working on this data matrix; this lambda n to the power half xn; you have seen yesterday also, you have seen many times; will give rise to what, 1 upper triangular component, multiplied by lambda to the power half. That upper triangular component will be Rn minus 1, then followed by a chunk of zeros, and last row will consist of the new data, x 1 n x 2 n dot dot dot dot xmn.

On that, this will work on rotations, so that; the last row will be finally annihilated, new Rn will be formed. That was the thing. In practice, you do not have to compute this Q at all. You just have to carry out the rotations; m rotations you have to apply. Assume, when doing this rotation and this part, I call as Tn, you have to apply this thing here also. Here, it means, here also the same Qn. So, Gmn dot dot dot G 1 n, into this Q bar n minus 1. This will work for this. This will work for this. This will work for this vector and out of which Q bar n minus 1, while working on this, will give rise to what, two components.

I mean, it will give rise to this thing, this part. This working on this, will give rise to Gmn dot dot dot G 1 n, into square root lambda upper part, square root lambda lower part and that last component. Q is not required. I know the upper and lower part for the previous index, just multiplied by square root lambda, put the new l guys, apply the same sequence of rotations. First, I have to carry out this. Then, from the upper, the first m components, that is m into 1 part.

I will take that out separately. Once I do this matrix stuff, the upper m plus 1 components will be the new Un, that will be taken out separately, and here the triangular matrix I have to solve that, and get the optimum filter coefficients. That is the business. Now, to do that, I will take an example. Yesterday, I was doing the systolic implementation, but it was half way, you know that there are some issues involved. Just to do that, let us take the case of m equal to 4. I took m equal to 3 yesterday; I thought m equal to 3 is too less, so m equal to 4.

(Refer Slide Time: 05:01)



We added index n, m equal to 4; we added index n, and also we will have operations like this. You will have some processes like this. First, these processes are suppose to rotate towards some data x, and a fellow comes u, then it will generate 2 components, c and s. How? Yesterday, I told you that there are 4 possibilities. One is that here, both x and u is positive. We rotate it this way. In that case, c will be positive; cos theta cos minus theta same, but s will be in this direction, if you see within that rotation matrix, s will come out to be positive.

Actually, c minus s, but since, it is in this direction cs; that is all that convection. On the other hand, if it is here, again, theta will be in this direction c will still be positive as equal, but s will be negative. Again here, if you rotate only up to this, c will be positive. Positive sends positive and s will be, because you are going in this direction, s will be negative. If you are here, you are going in this direction, c will be positive, as it is, but s will be positive. So, you see c is always positive and how much is the c value, you take this given x, take the mod. Find out this length, take the mod and take the ratio.

That is the value and give a sign to that, equal to the sign of the x. Yesterday, you wrote those mod x by, I mean, those things are not required in hardware. These are more complicated. Just sin of the number, you know is very easily found out in hardware. Here, we take the sin and take the sin we do not need.

This book is in old style; better not to use those kinds of things. So, what it will do and that special case, wherein this u is 0, that time no computation is required, that case we

take out separately. Otherwise, the remaining part also takes care of that, but why do the computation, knowing this is trivially offset. That is why I said for U equal to 0, c is obviously 1, s is 0, and this x will be replaced by x itself. No need to do any rotation and all. So, x goes to x else, what happens? You find out x prime. X prime is you have to bring in this triangular part, that element. This, like you understand where from, this elements comes; that is the upper triangular part, so you consider the past entry, that and u, the last row. Naturally, when I do the rotation, first, the square root lambda comes here.

Square root lambda terms is this x and then, 0 0 0 and then, u. So, when you rotate you find out this new length lambda, then your c is, either you have x and u here, or x and u here, and x and u here; you have the n value. This will be a positive sign. Then, c is sin of x times, sin of x times x prime, I am already taking the past value is fine, and s will be, magnitude of s is always u by this, u by this length. So, that part I first write; magnitude u by your into.

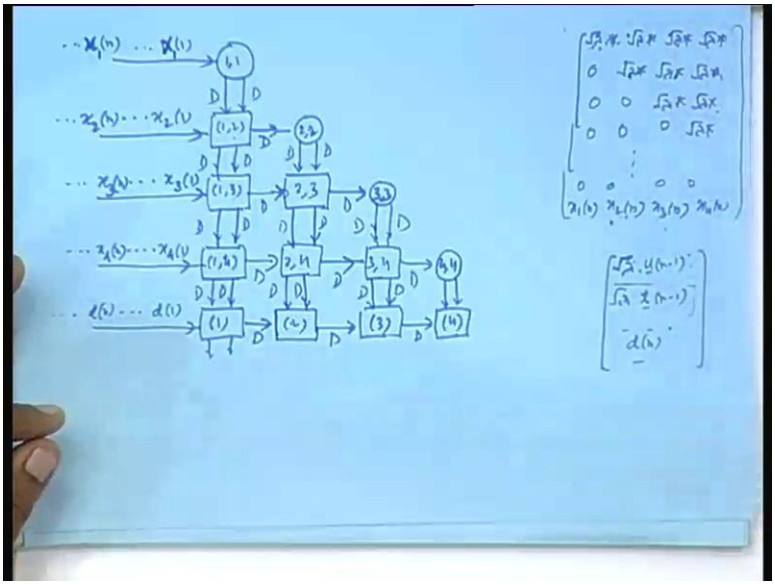
When you see, if you just count those cases, when either this and this are positive, or this and this, both are negative, then only s is positive. Otherwise, negative. This is the AND function actually. So, this is sin of, if you take the product x times u, when both are positive or both are negative, either case, sin will be plus. Do not think, in hardware you really have to multiply that 2 and data sin. Sin of ab is sin of a into sin of b, which is actually, a logical AND operation. I am not writing all that. This, you know, this is an elementary hardware. This rotates and then, s will be replaced by x prime; this is one processor.

There is another processor. This kind of processor here, if it is, take cs, they will propagate through it. Same cs will come out. Along this line, no processing. If a u comes and if you store some value x, u prime will come out and x will be replaced by x prime. What are those things? What does it do? What is u prime in terms of all these? U prime will be; u was that last row entry, you are rotating; you have found out cs parameter, using that, you have to take any other column on that Rn, that matrix, you understand.

In first column, we annihilated the last row; last element and that becomes 0. You got a c and s parameter, after you apply the same rotation on the other columns also. So, those elements, in those columns, last row is this u, and top row element, first row element is here, and that has to be rotated. This will be multiplied by square root lambda

So, output will be c square root lambda x, sorry, not c; this is x prime first; x prime will be c square root lambda x plus su, and u prime will be; that cos and sin and minus sign; minus s and x prime goes to x. With this description, suppose, I am dealing with the case, m equal to 4, as an example, capital M equal to 4 with this description.

(Refer Slide Time: 12:51)



I do like this, I will be writing within bracket 1 comma 1. Our starting index is n equal to 1; not 0; n equal to 1 dot dot dot dot current index un, sorry, it is. I am still not doing the filtering case. This is more general case, where m different signals are there; x 1; x 2; up to xm. In the filtering case, other things are just delayed versions of this line. I can afford to you, generally this is linear. What are these things, you know? At the n min at the nth index what you have?

You have 1 upper triangular matrix that is square root lambda times Rn minus 1, followed by zeros and then, last row, isn't it? You have this thing, square root lambda times, I am not writing this element. This will be stored here, upper triangular, then dot dot dot dot 0s and then, last is x 1 n x 2 n. What I mean, this guy, look at this matrix 1 comma one th element; this is stored here. 1 comma two th element; this guy will be stored, and it will after rotation, update this value also. I will get nth index, those values, this value is stored here; 1 3 1 4 is stored here. Similarly, there is a d, what is there in that d? Square root. What is this d after all? Square root lambda, i am using those notations only; i rewrite what i wrote earlier.

On this, I have to do rotation, like here also, on this I have to do rotation; here also, on this I have to do the same rotations. Rotation will be derived from this matrix and also applied on these.

When I say 1, basically, the top entry is stored; top entry of this vector. I am not bothered about this. No need to store this, why? Because, I always bothered about the first m cross 1; that is 4 cross 1 component, because only that will be used in a solution, for finding out the filter coefficients. Here, I am not bothered. Here, I got 4 entries. So, there also we have 4 places to store this; top guy and then, there will be other fellows also. What this guy does? At this moment, no pipelining, nothing. Everything is done all together in 1 cycle. if everything done and then, again new data uf data front, I mean data, you know, vector moves in that way.

What this guy does; obviously, it computes the cs parameter, you all know and I have given the algorithm. It will give the cs parameter ci, and that will propagate, because that is how this process are, whatever comes through, they propagate like this. Here that propagates, but no use of this. What does this guy do? We apply the same cs, on what, this fellow; that is this and the new guy that is entering; rotates them so that, what comes out? You are applying the rotation on this fellow and this fellow now. Rotation cns derived from these 2 applied on these 2, out of which this will get updated to some value, that will come back here.

Please note this, that will come back here, and this will get updated to some value that will come out. Why I want to store this? These and these after that, c times these and s times these, and it gets changed to some value. Why I am also bothered about storing that here? It is because, next time at n plus one th index, when a new row comes; x 2 n plus 1 will come. Again, new c and new s will be applied on this. So, this rotated value has to be stored. This value as to be annihilated, but this value, modified value has to be stored for use in that n plus one th index.

So, that is getting stored, and this new guy is coming out. That new guy and this fellow are to do the rotation, so that is annihilated. This will then, go to another fellow. This will be storing which element; 2 comma 2 diagonal elements are stored. This element is stored here in this processor. With that element, under the new value of this, it will compute new cs parameters.

One 3, this rotation was applied on these, where you take this column; even in this column, this fellow and this fellow are altered. This new value is stored here for future usage; n plus one th index. But this altered value comes out here, that is to be rotated by new cs, that comes out of these 2 guys. That will be applied on these, this will be applied on these; c on this; s on this; say one rotation or two. C on this, s on this, minus s on this and c on this; that is what is done here. So, these guys store these elements, 2 comma 3, and why only 2 comma 3?

You are applying this rotation on these 2, you have to do on these 2 also. Out of which, after the first rotation came out of these, this value got modified and got stored here. What came out, on that the new rotation coming from these 2, will be applied on these and these. So, this is stored here. Same cs will come. Here, 1 3 2 3, that is, this was rotated, then these 2 are rotated, then this fellow, these and these, this needs to be annihilated. So, this comes out. This value is stored here.

Out of those 2 new c and s come out, this is annihilated. That will work only on these 2, isn't it? Only these 2. This value will be modified and stored here; this is 3 comma 4; third row fourth element. And this value will come out. That and this will be rotated, so that, this is annihilated. Now comes to this d, first set of rotation 1 cs parameter work on the last guy and first guy. First guy stored here, last guy enters, rotate. This is simple hardware.

So, first guy modified value comes here, last guy moves here, then first but 1, second guy and again, this new last guy; second guy was stored here. Second guy and the new last guy; on them this rotation modified second guy stored here, and new last guy comes. That and the third guy; on these rotations, modified third guy gets stored and new last guy comes out on that, the last rotation. Understand all these values, that are stored down, they are important because, these values will give rise to that upper triangular matrix, after the rn. Rn will be content now, after this entire operation, all the operation is done; this part will consist of rn, isn't it? This I am telling you lower triangular, because of my drawing, but you can easily identify the limits rn, and this will give rise to that upper part, which is to be solved.

Solution is not done, but it is done separately. Because, it is an upper triangular solution and that is done, again by an array; linear array that we will come to. But, this is now

those who have done, this is just for triangularization on the matrix and corresponding rotation of the upper part of this vector. How will you get that?

Now, here as such, this computation you have to do; followed by this; followed by this; followed by this; followed by these two. You see, if I have to do the entire computation, so much time will be taken. Before I allow the n plus one th data, vector 2 be, I have to complete these; that result will be here, to complete this; both this; then, this; you can find out the total time required, that will be quite huge. But, those who have done my VLSI for telecom course, or those who know pipelining all that, at least in principle I can bring down the complexity very much here.

Do you have an idea about pipelining? Pipelining is like, suppose, there is a pipe; I am throwing stones. This is how I taught these. I am throwing stones through the pipe. Each stone takes some time to set out; to come out of the pipe. Once it comes out of the pipe, if I send the next stone, then I am throwing stone only at a rate of tau. But suppose, I say that I would not wait for the stone to come out of the pipe. Let it go little bit and then I will throw another one; then little bit and another one. So, my rate of transmission will be higher. So, instead of pipe, it is a total process and instead of stone, it is data.

Instead of waiting for the data to be completely processed and then coming out, and then sending in a new data, let me send one data and let it be processed partially, and then passed on to next stage of processing, and then I give in another data. That means, I divide the entire process into small processors; for synchronization I keep latches. So, that 1 cycle; each of them takes time less than my clock cycle, otherwise, there will be conflict. But I give one cycle to it or less than one cycle, it does the job, only after the cycle is over; passes on to the next line and so forth; that means, I have to break that job into small sectors and I introduce latches; pipeline latches

Here, I do not have any break; these are already broken. Suppose here, I put a d; d for flip-flop; one d, d d d. Those who have done my course, they know I will apply cutset here, for that possibly, you know more than me here. D d d, again here, d d d again here, say I am going horizontally; not vertically, I am going horizontally; that means, in one clock cycle, these operations will be done; this followed by this; this followed by this; this followed by this; this followed by this.

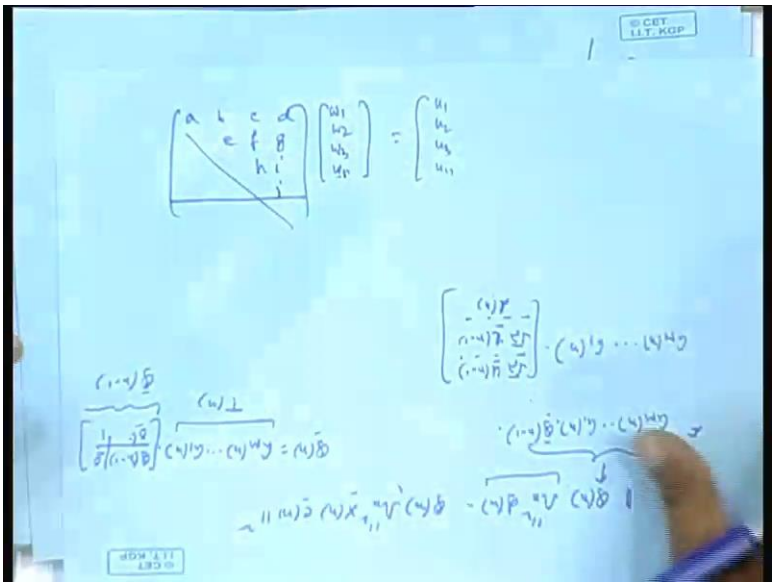
Once this is done, then you pass on the stuff in the next. When it processes this result, you can accept the new data; x 1 n plus 1; x 2 n plus 1; like that. You do not have to wait

for the entire thing. So, it may come down. I can go vertically also, where the same logic, then you taken as a home task to see the sequencing of data; how the data moves through the array.

Here, when it was only horizontal, you know one cycle all the processing then, next cycle then, that output moves here; x 1 n plus 1; x 2 n plus 1; they come together; they will be ahead. It will be delayed in both ways. So, that sequencing, the data movement you just have to work out; which data, when and where and all, but d d. Connect this, we will not be affected, you can verify that. That means, my clock will be (( )) by what, which one of this processor takes maximum time because; all processors are pipelined; these latches are there, between every two processors; whether of this kind; or this kind or between them.

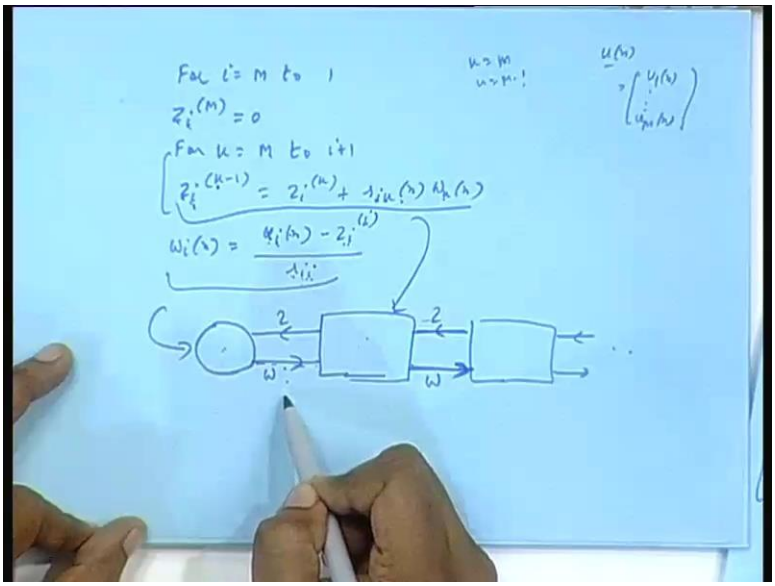
Apparently, this portion takes more time because, in the square root computation multiplication and all that. So, if it takes tau second; tau millisecond or micro second or whatever; that will be the determining factor in my clock. My clock period just has to be tau in the minimum, or it can be greater than tau, that is all. Those who know my course, they know I am applying cutset here, and then, I am applying cut set here. That is very obvious. Those who have done the course earlier, will realize what I am telling. For them, I am saying these are followed path and applying cut set. That is one thing. Back substitution, you know it is just written in algorithm manner, but I will explain, I will just mention from the book.

(Refer Slide Time: 30:37)



What is back substitution? That is I have got a triangular section say, a b c d e f g h i j and then, 0s and say, w 1, w 2, w 3, w 4. 0s, I do not care. This part and this part, equal to say, your upper matrix u 1, u 2, u 3 and u 4. What I do, first find out w 4 by solving the last equation, then propagate back using the same w 4. For the second row, i into w 4 not known; find out the h and then, w 3, w 4 known. So, propagate back, f into w 3; d into w 4. Find out; subtract and find out w and so on. So, it is a back substitution. Again, this can be done by an array and it is linear array.

(Refer Slide Time: 31:49)



Instead of working out, since time is less, I will just write it down from the book, and explain to you. Suppose, I have indices anyways like this; for I first write down; we go down; m to 1, m minus 1, m minus 2, like that. This is given starting value 0. You just write in an algorithm manner. We will have it verified; instead of defining an algorithm, I am just writing and explaining because, time is short. I have to end up today.

This is 0. Then for k equal to m to i plus 1. That means, when i equals to m; you do not execute the loop because, m plus 1, sorry, just a minute, let me write down first. Then, that statement will be; this u is for that upper un, ln that upper part of the matrix. You remember, un vector and ln vector; I am bothered only about the un, and un; the components will be u 1 n to u m n. This is how I will write the components. So, u i n, z i i where by, r i i. Let us see what it means. Suppose, m equal to 1; i equal to m. So, z mm be 0, and here, k equal to m to m plus 1. Let me see whether I am writing these correctly. I will come to this later.

W i n w m, last component, I am coming to this later. W m n, you see, it is like u m; last component here, minus z m m, that is 0 divided by simply r m m. That is the last equation; last equation r m m here, and u m here, by r m m wm. That time, this loop should not be executed. I do not know whether I am, this is only executed. Yes, good, we are going in the negative direction; I have got k and i, both are going from m to 1. So, next time, i equal to m minus 1. This time, I know, i equal to m minus 1. I am here, this is not 0; only when i equal to m, this is 0.

You consider the k. k means what; m to m. So, only one loop and that times, z m minus 1. What is k; m that is 0, and this is that m minus 1 m n, into the new value of this guy; m minus 1 is through n th column; that guy into this, and this is 0. This part, I am giving it name; z m minus 1, m minus 1, both of these two are same. Then only, I will bring back here. And that comes u m minus 1 n; that is last guy, minus that part, divided by this component r m minus 1 m minus 1. That will give you this. Then, you go for i equal to m minus 2, i equal to m minus 2; means k equals to m; then m to m minus 1; k equal to m case; then k equal to m minus 1, is it not?

So, twice, k equal to m means; after that we will clear; k equal to m means, this is 0. Whenever, k equal to m; this top fellow is 0. Because to the right of this, there is nothing; there is 0, which is why. This is 0, the starting is m here; m means, this corresponds to the column; last column, m minus 2 th row, last column; that entry times first guy; that weight; m th weight. That will be given a name here, what name; m minus 2 and m minus 1. Then, that will be brought back here; this k is now first m, then m minus 1; that will be brought back here.

And, this is one column to the left now, that was n th columnm now n minus 1 th column; 1 column to the left, that times n minus 1 th weight, that with this. Then, it becomes z m minus 1, m minus 1, which you bring back here. U m minus 1 minus this by, this is how it is. Actually, this is like writing the algorithm you know. Physically, you know how to solve this; this is just algorithmic form. This is done through a linear array. I will just again do like this, schematically. Actually, what is done is, here, it is the backward forward movement.

I mean, in the triangular section, everything was forward direction; no feedback. But here is a feedback kind of thing; feedback, because of the back substitution, that is the feedback thing. That is why, back substitution. Weights will move in this direction. What

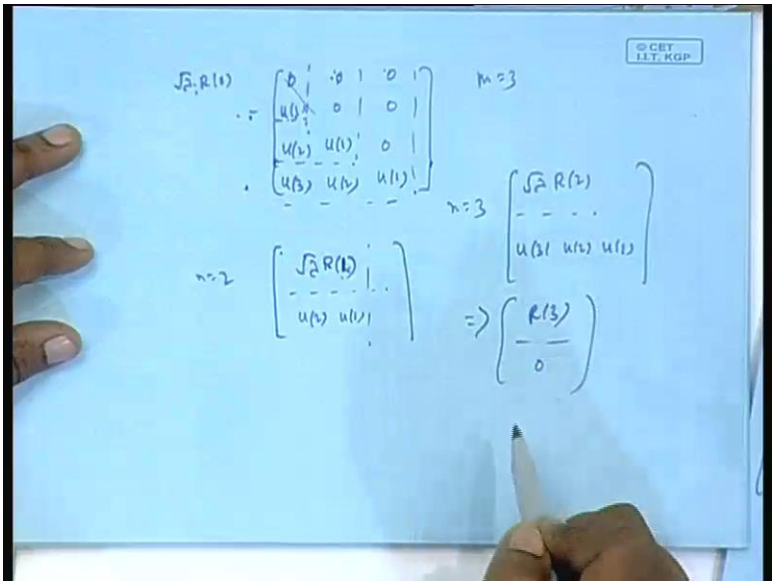
this guy does is, it will generate weights; this weight. I am just drawing the schematic. This we will do; this operation will be done here, and this operation will be done here. Using exact variable names and all that, you do but, these operations will be done here and these operations will be done here. In the very starting index, you know when i equal to m; this is 0, you are finding out and this r values are stored. They are available; r values are stored; r m m is stored here; using r m m and u m m, this guy first find outs w 1 m.

W 1 m and from here, that time 0 comes, w 1 m and this guy has stored, or m minus 1 m; that into this, you have to do this equation. You can do it yourself because, I mean I have to complete that other part; that linear dependence, independence thing, that for n less than m, whenever all 0 columns coming, you remember, that part also has to be completed. Can you quickly see this? Here, the weights will move and here zs will move.

New z from old z will be computed here. This z computed, becomes z i comma i then only, it reaches here. That minus the stored value, subtracted from the stored value, divided by the stored value; that will generate new weight. I suggest you, work it out. Because, for another 5 minutes, I do not have to take a class. In that class, you know, remaining 45 to 50 minutes will (( )) that is not something good.

Something, I have to wind up here. Only if time permits, I will come back to this. Coming to that issue, where I had got n less than m. That time I said that all 0 columns will come up. And therefore, QR, that way it is possible because, my first assumption on QR factor is, that the columns are linearly independent then, therefore, you do Graeme – Smithorthogation, you get orderly mutual orthonormal vectors, and then go outside the space and take few more orthogonal components. You get a full or unitary matrix, Qn. That is what I said. When the columns are not linearly independent, you do not get that. What to do there, remember, all throughout the derivation, I have not changed the order; order was fixed. You move from nth index to n plus oneth index; this m never changed to m plus 1, n minus 1. But here, what I will do, at n equal to,1 see my matrix.

(Refer Slide Time: 41:11)



I will write like this. 0 my data starts at 1; this n equal to 1; and you now consider; I will be considering this particular case linearly dependent or independent for the filtering case. Filtering case means filtering with pre windowed. Pre windowed means, whatever will be the first column; z inverse of that will be second; z inverse 2 of that will be third; and likewise. As you did that recursively co lattice and 1 is my starting index; 0 will be there on top always. Unlike there, here my starting index is n equal to 1, like that, 1 2 3, say m equal to 3 we have.

We have m equal to 3 case. What I will do, at n equal to 1, I will not consider these two columns. My purpose is, to find out that n equal to 3, I should be able to get that Rn matrix at n equal to 3. Because, my final m is 3. At n equal to 3, that is realized and after that n moves, but m does not change. I will not have that problem. So, n equal to 1, I consider only this part, although this matrix is given to me. This I view as square root lambda R 0; no problem, R 0 is my 0; this and this, I do rotation. If I rotate, this will move up, 0 will come down. Then what I get, at n equal to 1, I only rotate these two, that means, this square root lambda 0, u 1 moves here, 0 comes here; at n equal to 2 then, what is the situation; at n equal to 2, I will take 1 more column. Remember, always number of row is greater than the number of column, and I am not having all 0 column coming and all that.

Here it was this much, and here it was this much. So, that is independent. The dependent's problem, which is coming because, number of rows was less than number of

column and all 0 columns was coming, that is not. All 0 columns in the case of filtering, I mean, otherwise, number of row less than number of columns and therefore, dependents coming; that is not present here, isn't it?

Especially, with this kind of set up. These two are linearly independent; u 1 is non 0; I am assuming u 1 is non 0. So, this cannot be written as a multiple of this. This I knew. That is why I am working. Obviously, first I take this part, one component; no problem. Then, I take this much, 2 components, but clearly, they are not dependent. This cannot be multiple of this. This is non 0; u 1 is non 0, assume. So, this time, what is the structure I have got; one data here; one data, 0 here, 0 0. These two 0s put together and data here, that will be like, 1 R and that, I will view to be square root lambda R 1.

That is, I am taking this much now. So, that will be what; n equal to 2, I have got square root lambda R 1; that is Rn minus 1, followed by these two; u 2, u 1. How many columns? Two columns only. This upper part 0 0 here, u 1 moved here, that part I am calling square root lambda R 1 and u 2 u 1. This I can annihilate. Dependents do not come; just two rotations will be required now. Here one rotation is required; two rotations will be required.

Again, 0s will form and this thing will come up; upper triangular matrix. I have to multiply that like, here I multiplied by square root lambda; that time I have to multiply by R 2 by square root lambda and I will take this much, at n equal to 3. So, till that n equal to m situation arises, there is an order updating; order wise movement. Do not take all the orders at the very starting index; that is the key. So, the dependence is avoided. At n equal to 3, I take all the 3, but I already have up, is this; these three 0s, 1 upper triangular fellow, these 3 zeros and this part.

Multiply the upper triangular part by square root lambda R 2, followed by u 3, u 2, u 1; now three. Again 4 rows 3 columns and; obviously, you know this linear dependent is not there. This is not a linear combination of these two; I mean we multiply the 3 by c 1, c 2, c 3, add; that they cannot be 0 altogether. This you have seen in that earlier lattice case also. Obviously, again I can rotate it; I can triangularize it. This can be annihilated and I get R 3; this gives rise to R 3 0s.

From now onwards, there is no problem. Because, I have made the situation, where n equal to m. M will, I mean, after that, new and new rows will come up and therefore, dependence will go; possibly the dependence. That is all for this course. I will take two

more lectures, but that will be not for you, because this is 38. Two more lectures; appendix 1 and appendix 2, which will be on elements of estimation and solution. So, those who are the PIG students and I want to know little bit of estimation and other things. I can tell them, sometime may be during the same or after wards, I will take two classes; special classes, not for this course, I mean not for study purpose, but this video recording thing.

That will be something on estimation theory, you know three basic kind of estimations ml maximum (( )) m squared and map, and what is the best estimate; best estimation is something, what is called, which richest, (( )) I mean error variance which is (( )) what kind of estimates lead to that (( )) When does our mean square of this; all adaptive filters basically, are mean square; least square also mean square. When does this give rise to your best estimate? In fact, when everything is Gaussian then, this will be the best estimate; that some treatment request to be given on one lecture, which I will give. Another lecture, I might give on some other forms of LMS. That, I will let you know, if you are in touch with me. How do I know, how to reach you out.

Anyway, thank you very much for this.