DSP: Transcript of lectures and recordings

An Open-Source Summary

AUTHOR

KU LEUVEN

September 25, 2025

Contents

This document is a speech-to-text conversion of DSP classes.

1 Chapter 4

1.1 Slide 3

So step one is really where to define the filter specifications and should give an idea of what this is like. Here's an example of low filter. Low filter is obviously filter that passes low frequency contents and attenuate stops high frequency signal. Which you see in the graph is a plot of amplitude response basically of the low pass filter. On the horizontal axis you see radial frequency the vertical axis you see amplitude response. So the filter design here has a passband so frequency range

where frequencies are basically passed by the filter that runs up to a radial frequency in the neighborhood of two and then a stopband that starts with the neighborhood of the same radial frequency which is 2. The filter is never going to have a ideal response which is basically a response that is flat at 1. So it's going to ripple about this one and here you can define a passband ripple to the maximum allowed ripple in the passband that after the passband that you're running should be transition bands. Transition bands should start at for instance a passband frequency or cut off frequency and then the transition should end at the stopband frequency. Here again you can see the attenuation should be we should minimally be equal to a certain value which is delta S here which is defined as the stopband ripple.

1.2 Slide 4

So once you have defined the filter specifications as you had in the previous slide the idea would be to design a transfer function that has a magnitude response something like what you had in the previous slide. So this is where we're going to be doing in this chapter and the chapter has basically two parts. The first part will consider FIR filters and the second part will have a similar story for IR filters. For FIR filter design turns out that linear face responses important and then we'll have FIR design which will turn out can easily be defined as an optimization problem with a least squares optimization problem or minimize optimization problem or whatever so that once you can treat your design filter design problems as an optimization problem then typically you have standard optimization software to solve problem and that is at least for IR filter design. It's a bit of a general approach that you can present here. Next to this optimization based approach, there are many FIR filter design procedures available and also used in practice and commercial software for instance and will get an idea of what these procedures are and how they are sometimes related to the optimization approach. Then in the second part of the chapter we'll look into IIR filter design which is bit of a similar story. I will say a few words about Poles and zeros placements to in a sense define the filter response and then make an attempt for a similar statement in that you could do IIR filter design or states or phrase it as an optimization problem and then solve it. The statement has been weaker here the optimization problems that you run into when you do IIR filter designer more involved or complicated compared to the FIR case. Then again there are many procedures and practice in textbooks and practical commercial software and also give you an idea of what that is.

1.3 Slide 6

The important aspect of FIR filters is that FIR filters can have a linear phase response which will turn out to be the desirable phase response. To get there in this slide we first considered what is referred to as a zero phase filter. So we'll consider FIR filters that zero phase response but in order to get there we will have to allow for noncausalities. So we will consider the first noncausal which are not for the practical indeed FIR filters. In the next slide we're going to take away this noncausality. Which you have here is the plot of impulse response of a FIR filter. You see nine nonzero tabs in the impulse response and vertical arrow indicates the position of K is equal to 0 so you see the nonzero impulse response coefficients both for negative time K and for positive time K. Nonzero impulse response coefficients for negative time K really mean that the impulse response already starts before the impulse actually happens right that in practice especially when K represents time this is a bit difficult to envisage. But anyway so let's consider a noncausal finite impulse response system and an example is plotted here. And we consider such an impulse response that also satisfies symmetry property. So with this symmetry and the impulse response from there you can inspect the frequency response of such a system. To get the frequency response you take the Z transform . The summation over all the input impulse response coefficients becomes a summation over roughly half as many terms and then all the terms rather than complex exponentials for cosine functions. So what you observe here is that because of the fact that your frequency response is a sum of cosine functions and then also inspect what the K coefficients are they're basically related to the H coefficients the impulse response coefficients

which have been assumed here should be real values so that decays are real valued cosine functions are real valued you conclude that basically everything in the frequency response is real value. So the frequency response is real value which means that it's basically the only amplitude and it doesn't have any phase or the phase is zero everywhere and hence such a filter is referred to as zero phase filter but then remember it has to be a noncausal filter od be able to come up with such a property.

1.4 Slide 7

A noncausal system is not very practical you can always turn the FIR filter from the previous slide to a causal filter by just shifting the impulse responses as a whole towards positive time K. This is indicated in this slide so in the in the graph here you see exactly the same shape of impulse response the same impulse response sequence but it's shifted towards positive time K so it starts at K equal to zero and then you have the first half of the impulse response and then the second-half which is the symmetric version of the of the first half. You shift that noncausal impulse response into a causal impulse response you will see that zero phase filter actually is turning into what is referred to as a linear phase filter. Check the frequency response again you will see that basically the frequency response is the same frequency response as what you had in previous slide it's the summation of the cosine fractions which is real valued and then gets multiplied by the colored part which is basically that you're evaluating a pure delay. And that face response is a linear function of the radial frequency Omega. I should also say that the linear phase response is a very desirable phase response. The zero phase response as you added in the previous slide your frequency just want to response was basically only the amplitude which means that once you can see would be amplified at another frequency would be attenuated for instance by your filter but not the frequency components and really get delayed or shifted there's no phase response to the frequency response in the previous slide. And this slide you have the same effect with a filter would amplify a certain frequency and attenuate a certain frequency which sort of matches which with your intuition of what a filter should be doing and on top of that basically all frequencies shifted over an amount of Lo samples I should say it would introduce the same delay for all of your frequency components. So linear phase desirable phase response because it matches with your intuition of what a filter should be doing basically it’s attenuating some frequencies and amplifying other frequencies while not sort of shifting frequency components relative to each other.

1.5 Slide 8

The example in the previous slide has a capital L which is basically the order of the FIR filter and that example the L was an even value and also we use symmetry in the sequence of response coefficient and that leads to a certain frequency response as you have it here also listed under type one. So type one corresponds to a filter order even and a symmetry that impulse response coefficients lead to the first yellow formula as we have derived in the previous slide. It turns out that if you use this expression with this you can with this type of filter you can do low pass, bandpass and high pass, whichever filter you would like. Now next to this type as it is referred to type one, you can also have different type of linear phase filters. The order the capital L in type one was even but it could also be odd and the symmetry can also be an antisymmetry or one filter coefficient is equal to -1 times another filter coefficient and then all together of course you have four possibilities with L either odd or even symmetry being either a symmetry or an antisymmetry and has four types of linear phase filters. When you look into the details for instance type two filter you would have to do derivation similar to derivation in the previous slides leads to a frequency response which has that second yellow formula here under type 2. It has an additional factor cosine of Omega divided by two and there you see that from mega is equal to Pi basically cosine of / 2 is 0 and Omega equals to pi correspond to the Nyquist frequency. So this type of filter will always have zero at the Nyquist frequency and hence you can never use that type of filter to build a high pass filter you can use it to build a low

pass or bandpass filter but never high pass filter. Similarly for type three filters type three filters will turn out you have to do the derivation again. Type three has a zero both have radial frequency Omega is equal to 0 and DC as well as at the Nyquist frequency. type three can only be used for design of bandpass filter and that type four can be used for design of high pass bandpass filters but never a low pass filter. A few things you can check here if you for instance take type 2 filter and you modulate impulse response with the sequence of numbers +1 - 1 + 1 - 1 which has an effective basically shifting the frequency response over the amounts pi you can check that type 2 filters then turn into a type 4 filter so L is odd in both cases and symmetry is turned into an antisymmetry and indeed you will also see that type 2 filter can never be a high pass filter because the modulation you basically shift the response by amount of pi so if type two can never be a high pass filter then type 4 can never be a low pass filter so that all starts making sense then. Important remark also is that the observation is that by introducing basically symmetry or antisymmetry in the impulse response sequence of a FIR filter you can impose the linear phase properties. IIR filters can never have this linear phase property and you can prove that. This is an important properties so linear phase property is something you can have with FIR filters which can never happens with nontrivial IIR filters I say nontrivial because in a sense you can also view FIR filters are special cases of IR filters.

1.6 Slide 9

Let us now see how you can face the FIR filter design problem as an optimization problem. We have here is a weighted least square filter design and as you go on you will see where the name truly comes from. So we are designing FIR filter and from previous slide we remember that linear phase response is desirable phase response so we are going to design linear phase response filters and to keep things simple let us focus on type one filters. So anyway your frequency response has a phase factor times an amplitude factor. Now imagine you are wanting to design low pass or high pass or whatever filter something similar to what you have drawn in slide 3 go back to slide three that would correspond to a desired frequency response that you can also specify as a phase factor times an amplitude factor. The phase factor can be the same with linear phase factor because we think that linear phase response is a desirable response and then you have the desired amplitude response which is denoted here as Gd. So way to do this is to define an optimization criterion and the optimization criteria will be quadratic criteria. Basically says we're going to tune the d coefficients such that the following expression is minimized and the expression is the integral over all frequencies radio frequencies from minus up to of and then you see W(w) it's basically your weighting function which introduce to give larger weights as to some of that frequencies and a smaller weight to other frequencies or frequency ranges for instance. The important factor is that the square of the absolute value of the difference between the designed frequency response minus the desired frequency response. So you inspect the difference between H and Hd. Now of course that H and Hd both have linear phase that factor which can be taken out from the equation. Last part of the equation is basically where you minimize over that set of coefficients so you search for the optimal set of d coefficients to minimize this lease square criteria or you basically check the difference between the G the designed amplitude response and Gd the desired amplitude response.

1.7 Slide 11

Simple example is provided in this slide basically goes back to slide 3. In sense with desired response for a low pass filter. So the design procedure will be something as follows here you would say that my desired amplitude response Gd in the passband would be equal to 1. The optimization function would then be the sum of two terms one term corresponding to the passband one term corresponding to the stopband and the pass band basically you compare the designed amplitude response G(w) with the desired amplitude response which is equal to 1 and then the second term is similar term for the stopband. The gamma is representing the weighting function. You can tune the gamma the way you wanted would like to

put more emphasis on the stopband or on the passband.

1.8 Slide 12

1.9 Slide 10 has a closed form formula for the weighted filter design and it

involved integrals so it has a matrix Q and a vector p. And the definitions of the Q and P have integrals and this is basically because also the cost criterion that you started from the optimization function and integrals here integrated over all frequencies. Now if you do not like integrals or you would like to have a more practical approach then you can replace the integrals by the summation over a collection of so-called sample frequencies. So if wi here are sample frequencies and the cost function the F function of the optimization variables d coefficients rather than an integral could be a summation over all sample frequencies and then for each sample frequency again you evaluate the difference between the designed amplitude response and the desired amplitude response and absolute value of the difference is squared and you can also include the weight function as you have it during the first formula. If you replace integrals by summation then similarly you're going to have definition for the Q matrix and P vector which have summations over all the sample frequencies but then again a closed form solution. So the optimal set of d coefficients is again the matrix Q inverted times the p which you basically have here for instance if you would have 100 sample frequencies is that you have an overdetermined set of linear equations and the d coefficients if you have 10. 100 sample frequencies basically each sample from Q matrix gives you an equation in the 10 unknown d coefficient and so the answer going to have 100 equations and 10 unknowns which is solved here you know these in a way that least square sense.

1.10 Slide 13

Design of the previous slide can be supplemented also with additional constraints for instance for the passband and stopband triple control. This goes back to basically slide 3 we have the desired filter magnitude response and we also defined allowable passband and stopband triples so the idea would then be to have additional sample frequencies sample radio frequencies a number of them in the passband a number of them in the stopband. And for the passband that sample frequencies you would specify that deviation of the designed response from one in absolute value should be bounded by the allowed passband triple which is the delta P and similarly for the stopband sample frequencies would say that deviation from zero to designed amplitude response and absolute values should be smaller than the allowed stopband triple which is the delta s. Now this would introduce additional constraints to your quadratic optimization problem the constraint with absolute value can turn into two constraints without the absolute value. So basically you're going to have quadratic cost function so this is the F as you have in the previous slides function of optimization variables d coefficients which are collected in vector x, and then the quality constraints corresponding to the passband constraints equality constraints corresponding to the stopband constraints all altogether this is quadratic optimization subject to inequality constraints linear inequality constraints. This is referred to as quadratic program or quadratic programming problem there's again software available for those of you who have had optimization course to solve such problems.

1.11 Slide 14

The way leads the least squares design procedure from the previous slides uses a specific quadratic optimization criteria and it's important to realize that this is by no means the only optimization criteria we can use here. You can use various other optimization criterions any criterion you can come up with is basically fine as long as you have procedure to solve corresponding optimization problem. So here's another example is similar to slide 9 so we're going to design an FIR

filter let's say linear phase FIR filter type one where the frequency response as the linear phase factor and the amplitude factor which is the way that sum of cosine functions where the d coefficients are the weighting factors. So that is design frequency response H which is the phase factor times designed amplitude response. Now we're going to compare that to the desired frequency response. Now rather than quadratic optimization problem what we have here as a second example is as a minimax criterion and the idea is as follows for each radial frequency, we will again comparing the desired frequency response with the designed frequency response so we're subtracting from the other to the absolute value squared and we could possibly also multiply with a weighting function and so this is evaluated for all radial frequencies. And of all these radial frequencies you picked the radial frequencies for which this expression is reaching a maximum and then the idea is to set the d coefficients such that this maximum deviation basically of the designed transfer function from the desired transfer function is minimized. So this is the minimax criteria again you can take out the sort of the common in your phase term and then you end up with the last expression you minimize over the d coefficients such that the maximum deviation between designed amplitude response and the desired amplitude response squared possibly weighted is minimized. It turns out that this optimization criteria so alternatives to the least square criteria and also leads to the standard optimization problem and the optimization problem here is where can be turned into what people refer to as a semidefinite program and for semidefinite programs there's again efficient software The message is basically you can define any optimization criterion that makes sense as long as you can come up with solution procedure to solve the optimization problem.

1.12 Slide 15

The intermediate conclusion here is that we have provided two examples with least square designed and minimax design which are two examples of representatives so to speak of a general framework where the design problem is basically translated into an optimization problem. If you pick the right optimization function then that leads to a standard optimization problem for which you have standard optimization software to solve your problems. In a sense you can say or conclude that FIR filter design you can formulate such that it leads to standard optimization problems and then basically your problem is solved. Now if you go to the library for instance and you check DSP textbooks on FIR filter design, you're going to see many other I would call them ad-hoc procedures to do FIR filter design. You can see design procedures based on what is referred to as window functions you would see something which is referred to as ripple design and many other design procedures. I refer to these as ad-hoc design procedures. So people have spent lifetimes to come up with good procedures for FIR filter design the next few slides I will focus on two examples window based design and then equiripple design. It's important or it's good to know what these procedures what they look like and then also to indicate how they are related if they are related to the optimizations framework as I had in the previous slide.

1.13 Slide 16

FIR filter design based on window function as explained in this slide and next two slide. It's a popular method that you will find in textbooks and also commercial software oftentimes. It's explained here by means of a very simple example of a lowpass filter design so if you're designing a lowpass filter the desired frequency response would be equal for instance one for all the frequencies smaller than the passband cutoff frequency and then zero for all frequencies larger than the passband cutoff frequency. So here's your ideal lowpass filter let's say and then you go back to your signals and systems courses and you're trying to identify what time domain impulse response this frequency response corresponds to and you will find in your signals and systems course that's the frequency response that you have defined here corresponds to an ideal time domain impulse response which you can compute or finding your textbooks which eventually is equal to a sync function as it is indicated here. Ignore the details but basically a sync function and you remember what the

shape sync function is basically one for K is equal to zero and then it triples and dies for large K both positive K as well as negative K. So the observation is that it's infinitely long impulse response and it's noncausal so it's nonzero both for positive K and for negative K but it's a function that dies. If you go to large K both large positive and negative K the value of that sync function will basically go to 0. So there you could take an engineering approach and you could say ok this is infinitely long impulse response and I truncated after L+1 samples to get a Lth order filter and set all impulse response coefficient to 0 and then I may still have a noncausal function which as we have seen it in the previous slides can be shifted to positive time. By introducing this shift you basically introduced that linear phase factor in the transfer function which is desirable. So procedure is you start from an ideal low pass filter in this case right ideal description of a low pass filter you transform that into an ideal time domain impulse response sequence which in this case happens to be infinitely long both for negative and positive time K but then you truncate it after a number of samples so you truncated to a finite impulse response filter and then you shift the impulse response to positive to end up with a causal filter. So this is a simple procedure and it applies to any filter designers as you can find any ideal filter as long as you find the corresponding time domain impulse response sequence.

1.14 Slide 17

The procedure in the previous slide is a simple one but at the same time it's been an ad-hoc procedure where you do not exactly know what type of approximation good or not so good approximation of the desired frequency response you're actually going to obtain. So it’s a straightforward procedure but whether it provides you with you good filter design is still up in the air. Now it turns out that if you do this procedure with the truncation as you had it in the previous slide that it actually gives you exactly the same result as if you were to apply weighted least squares design procedure with exactly the same desired frequency response. The weighting function in the way that leads least squares design procedure is set uniformly equal to 1 so that means that the truncation procedure from previous slide gives you a solution that actually is a weighted least squares solution for particular Hd and for a particular weighting function. And that is in a sense that good news. So where do the window functions then come in if you go back to the previous slide the finite impulse response has basically this sort of the ideal infinitely long impulse response which is truncated and the truncation basically corresponds to a multiplication with a rectangular windows so this is the W[k] that you have here which is which is one for the samples that that you keep them which is 0 for the samples that you throw out. So basically you first form the ideal typically infinitely long impulse response Hd and then you multiply it with a window function which in this case is indeed a rectangular window function. Again this is a very simple procedure you can apply the procedure for whatever filter design. A negative point here is that the type of truncation in the time domain you do here is known to result in what is referred to as the Gibbs effect which means that if you're approximating basically an ideal frequency response or amplitude response which is discontinuous and then at the band edge and this discontinuity is of course that passband has turned into a stopband at this discontinuity. After the truncation we're going to see a large ripple in your approximation both in passband and stopband and basically the ripple doesn't go away even if you increase the filter order. If you increase the filter order you keep more samples from the ideal Hd impulse response but the ripple will not go away and that is the very well know Gibbs effect. So that tells you that the rectangular window is you have applied it here may not happen to be the ideal window. Question is if you can do the same procedure but perhaps with a difference window function.

1.15 Slide 18

Question is whether you can do the same procedure but basically with a different window function different from the rectangular window function at the previous slides. Perhaps indeed you're better off with the window function which has

a smoother transition between the area where you keep impulse response coefficients to the area where you throw away impulse response coefficients. Compared to the rectangular window function which has a very sort of immediate step transition between the area where you keep the impulse response coefficients and the are where you throw away impulse response coefficients. In general if you if you apply a window function so in the procedure you first define the ideal to be infinitely long impulse responses sequence that's Hd in the first formula you multiply with the window function W[k] and so now the W can be different from the rectangle window as you had in the previous slide gives you the finite impulse response filter h[k]. Equivalent to this time domain multiplication you can see that in the frequency domain basically the ideal frequency domain or frequency response of the ideal filter gets convolved with the frequency domain representation of your window function and this is where people have spent so to speak lifetimes to come up with suitable windows. There's a large choice of window functions Basically the window design or the choice of the window basically leads to a trade-off between on the one hand large or small pass and stopband ripples versus very small transition band from the passband to stopband and this respectively has to do with side-lobe levels in the frequency response of your window function versus the width of the main-lobe of the frequency response of your window function.

1.16 Slide 19

Another FIR filter design procedure which is very popular in textbooks and commercial software is referred to as the equiripple design procedure and this is related to the minimax optimization problem that has been defined in in slide 14 which is repeated here for convenience. So remember we're evaluating for all radio frequencies the difference between the designed amplitude response and desired amplitude response. And the difference between the two possibly weighted with weighting function. This is valuating where this expression achieves maximum value for all radio frequencies and then this maximum value should be minimized so we're searching for a set of d coefficients that minimizes this maximum deviation between the designed and desired amplitude response and basically differences also defined here as E(w). Now in slide 14 I've mentioned that this can be turned into what people refer to as a semidefinite program and that's a larger class in a sense of optimization problems for which again you have software available so once you turn the specific problem into semidefinite program would basically be considered the problem as this as being solved. Now what we have here is basically rather than going to the broader class of optimization problems we have procedure here that explicitly solves this specific optimization problem this minmax optimization problem and the solution procedure is based on Chebyshev approximation theory that basically says that if you consider this minimax optimization criterion, the solution to that problem will be such that the maximum deviation is basically going to be maximized in a number of radial frequencies which are referred to as extremal frequencies. So if the number of d coefficients here is L0 + 1 Chebyshev approximation if you would state that there's going to be L0 + 2 extremal frequencies and in these extremal frequencies basically the deviation between the designed and the desired amplitude response will reach maximum. If you envisage this or draw it then you will see that basically you have designed filter amplitude response which will ripple about the ideal amplitude response and the ripples are basically all equally large and hence equiripple design. Based on that Chebyshev approximation theory you can do specific algorithm names remember here is the Parks-McClellan algorithm to solve this procedure basically by first computing the extremal frequencies and then optimizing the d coefficients. So that is a particular procedure to solve this minmax optimization problem and we're not going to go into further details. Message here is so what is referred equiripple design is basically relies on a very specific procedure to solve the minmax optimization or design problem as we had in slide 14.

1.17 Slide 20

Next to the procedures in the previous slides there's many other procedures available for FIR filtering design. Too many to discuss here the good news is basically that all of these procedures are commercial software packages and you don't need to know each and every detail of such a design procedure. This is a MATLAB filter design and the actual design for you reduces to basically one line commands like you have a few examples here. You would say B is the vector with filter coefficients that will be returned by the MATLAB procedure and there are a few procedures available in MATLAB.

1.18 Slide 21

The next few slides you see a few FIR filter design examples based on MATLAB procedures. What I have here first this is a bandpass filter design where the order of the filter is gradually increased and so we'll see in next few slides that as you increase the filter order you obviously get a better band bandpass response. So here's the first design result that when the filter order is only 10 and the bottom part of the figure you see the impulse response so it has 10 or 11 nonzero coefficients and you see that it's basically a symmetric impulse response it's symmetric about 0 which has that been shifted to positive time K. So short 11 zero coefficient impulse response frequency response is not much of a bandpass filter so the idea would then be to increase the filter on it to get better bandpass filter response.

1.19 Slide 22

We increase the filter order we are going to have a longer finite impulse response and you already see a bandpass filter appearing.

1.20 Slide 24

Finally we set the filter order to 100 we gat perhaps bandpass filter response, an even longer filter response. The MATLAB code produce these figures is included here. The observation from these last four slides basically that typically when you design FIR filter you start with that particular filter order and if you're happy with the results you're happy. If you're not happy you typically increase filter order until you are happy.

1.21 Slide 25

The next few slides we are comparing window based design for number of possible windows. So here in the first slide we have bandpass filter design which is based on the triangle window. In the bottom slide you see the resulting impulse response so this is 50th filter order that we're using and in right you see the window function that has been applied which in this case so is it triangular window. You see that resulting response probably not very good bandpass filter.

1.22 Slide 26

And you switch to rectangular window. This is what you obtain and bottom figure you see the rectangular window and the resulting same filter order FIR impulse response. Result from the window based design when we use triangular window is still plotted. What you see is with the rectangular window you have the passband, the transition band and ripples in the stopband.

1.23 Slide 27

This slide we use another window which is hamming window which shown in the bottom part of the figure as well as the resulting impulse response sequence. And on the top figure you see the frequency response to design using this hamming window together with the results from previous slides using a triangular and rectangular window. Which observed now with the hamming window again you're going to see ripples in the stopband and passband filter. The ripples clearly now set at the much lower level compared to the ripples when you do a design based on a rectangular window. That is better you could say. What you pay is that the transition band when you use this hamming window is much broader. When you go from the passband to stopband with the rectangular window based design you have a steep transition from passband to stopband whereas when you are using hamming window you have a much broader transition band.

1.24 Slide 28

Last example here we are using window which is referred to the Blackman window. Blackman window shape is again indicated in the bottom part of the slide with the resulting impulse response sequence. The interesting part is the frequency response that is obtained which you observe again compared to the previous designs what you observe is that the ripple in the stopband is lower but then what you pay is broader transition band. Transition from passband to stopband is broader than in the case of the hamming window and rectangular window. So what you observe is the tradeoff between the width of the transition band with the height of side-lobe level in the stopband.

1.25 Slide 29

Basically that FIR filter design is rather simple, either you could sort of follow that general optimizations framework or you first rephrase your FIR filter design as an optimization problem which you can use standards for optimization software. Next to that there's a multitude of ad-hoc I would say design procedures, we've seen window based procedures and there's many others and the good news is basically we don't want to hear all the details of these procedures. That's all very well documented in commercial software for instance in MATLAB everything is available and you can play with these design procedures. So next part is about IIR filter design. Question is does the story also carry over to IIR filter design. If we are considering IIR filters we're considering transfer function H(z) and a rational transfer function obviously is defined by or defined as poles and zeros. The poles have to lie inside the unit circle. A(z) unlike in the case of FIR filter is nontrivial polynomial will have infinite long impulse response and hence the name IIR filters are infinitely long impulse response filters are rational transfer function like you have it here. Response to the difference equations so you can spell out the difference equation and also say in the last formula that output sample y[k] is basically a linear combination of the last so many input samples where that weight in the linear combination is b coefficient plus or minus a linear combination of so many outputs with the weight a coefficient. The part with b coefficient is sometimes referred to as the moving average part. Part with the a coefficients is oftentimes referred to as the autoregressive AR part.

1.26 Slide 30

The message in the previous slides or from the previous part is really that FIR filter design is really simple then you could ask why we would be interested in design. Then the motivation for IIR filter design is apparent from the examples that I have shown in previous slides. On the examples you saw that you typically design a filter and then you're not happy with the results and to get better results you increase the filter order. You're happy with the result but oftentimes you end up with a high order FIR filter which also needs to a high implementation complexity. If you build the filter and go

back to the difference equations for instance as you have in the previous slide as you also had for the FIR case you would see that an output sample is a linear combination of the last too many input samples in the FIR case with the number of input samples that you have in this linear combination that's very large that you have to do many multiplications many additions to compute 1 output sample. So the runtime complexity is basically defined by the order of the filter so you should design. So the hope is that with IIR filters we can have much lower order filters that nevertheless produce sharp responses for instance. If sharp frequency responses are good approximation of ideal frequency responses in the frequency domain come with long impulse response as you have seen in the FIR case. And IIR filter no matter how low the order immediately has long impulse response so the hope is truly that we can design good filters with a low order and hence a low computational cost. Again go back to the difference equation in the previous slide to see that the complexity to compute one output sample y[k] is related to the number of coefficients that you have your IIR filter. So this is the motivation for looking into IIR filters, the bad news will be that IIR filter design is more difficult than FIR. For IIR filter design you also have to keep an eye on stability. Another difficulty is that in the FIR case we could have linear phase response which you can never had with IIR filter. And then finally you will see in a later chapter that if you go out realizing and implementing an IIR filter, because basically it is a feedback system as we'll see you could run into issues with coefficient sensitivity quantization noise etc. which would be much worse in the IIR case compared to the FIR case.

1.27 Slide 31

When designing IIR transfer functions it's good to remember that the frequency response as you're designing it is obviously a function of the location of the poles and zeros of the transfer function. So if you know that location of poles and zeros so you set the location of the poles and zeros you can somehow guess what the frequency response is going to look like. And frequency responses is of course the z transform evaluated on the unit circle which you have in the figure here is an example z transform which is evaluated inside the unit circle. So in the XY plane you see the unit circle with radio frequency you make equal to 0 for the dc component and then -1 for Nyquist frequency. And this example we have two poles which lie inside the unit circle and we have two zeros. All together it is a second order IIR transfer function. In the neighbor of poles H(z) goes to infinity so it's pulled up which means that if you place poles near the unit circle this is where you can basically organize passband response. So passband can be set in a sense appropriate pole placement. Similarly by putting zero close to the unit circle you're going to organize stopband. Stopband can be emphasized by proper zero placement.

1.28 Slide 32

Question now is whether we can for IIR filter design follow a similar procedure as we had for FIR filter design. First you define a optimization problem. Obviously specifying the optimization problem is not very difficult and there's an example in this slides an example in the next slide. In this slide we do basically squares design follow up procedures similar to the FIR case. So we first specify our transfer functions in this case the transfer function H(z) is a rational transfer function. We're going to compare that corresponding frequency response with the desired frequency response which is indicated here is H subscript D of radio frequency Omega and in the FIR case we would specify the desired frequency response as factorization into phase response and amplitude response with the phase response is linear phase response and then we could actually remove the linear phase response from the optimization criterion. In this case we're not going to do that because remember IIR filter never has linear phase response and so the phase responses such will be gradient of the design procedure. Nevertheless we can specify the optimization criteria minimizing over optimization variables b coefficients and all of a coefficient and the optimization function is the integral over all radial frequencies weighted squared difference between the designed and desired response frequency. This is our optimization criteria next to this

optimization criteria we also have to impose a stability constraint not any set of a coefficients is a good set of a coefficient. We should have a coefficient such that the A(z)_ polynomial is stable so we should have A polynomial which is the never 0 for the z outside the unit circle.

1.29 Slide 33

Similarly we can define minimax optimization criterion so same transfer function same desired frequency response and the optimization function as minimax optimization function. optimization variables are still the A and B coefficients and then you can minimize the maximum deviation maximum evaluated over all radio frequencies of the weighted difference between desired and designd frequency response and again we have to include the stability constraints.

1.30 Slide 34

As was mentioned in the previous slides defining an optimization criterion is simple then solving the optimization problem is more difficult and it turns out that indeed solving the optimization problems like those in the previous two slides is significantly more difficult than solving equivalents optimization problems for the FIR case. there are basically 2 additional complications. Problem 1 is the presence of denominator polynomial which for instance if you consider the weighted least squares criteria in FIR case that's very simple quadratic optimization problem. In this case because you have the denominator polynomial it's not going to be to the quadratic optimization problem. So it's nonquadratic nonlinear optimization so more difficult to solve with nonglobal optimum etc. Second problem is the additional stability constraints so you cannot pick just any set of coefficients. The a coefficients should be such that if you form a polynomial with these coefficients, the zeros of this polynomial have to lie inside the unit circle have to be stable. Now the zeros of a polynomial are complicated function. You remember that it's the 3rd 4th or 10th order or whatever order it's impossible to specify the zeros the polynomial as a function of the coefficients so that is complication because this stability constraint is unusual or difficult to deal with constraints of the coefficients. So I hear there are solutions that use alternative forms of the stability constraints where we use affine functions of the filter coefficients rather or as an alternative to these strange functions of that coefficients of the polynomial. The things to remember is formulating the optimization problem is easy but solving the optimization problems not saying it's impossible but it's more complicated than solving the corresponding or equivalent of optimization problems in the FIR case.

1.31 Slide 35

The immediate conclusion here is that you have similar to the FIR design case, you have general framework or you rephrase the IIR filter design problem as an optimization problem. Once you formulated your optimization problem you can try and solve this optimization problem only in this case unlike FIR case you would say optimization problems are not so easy as the optimization problems in the FIR case. These is no standard optimization problem like quadratic problem. Not saying that it cannot be solved that there isn't any software available to solve these optimization problems but it's definitely more difficult. If you run to the library and check DSP textbooks many ad-hoc procedures do to IIR filter design procedures are based on analog filter design there's a large literature available on analog filter design. There are results by the famous people here like the Butterworth and the Chebyshev etc. So these people have given us design procedures that now it's to design an analog filter but it turns out that such analog filter design features can also be converted or translated into digital filter design procedure. This is what you often see in commercial software package and what is often used. There are many other procedures available for design based on modeling system modeling procedure.

1.32 Slide 37

The final slides here show a bandpass filter design again based on a Butterworth procedure. And go from one slide to the next you'll see that order of the IIR filter increasing. So here's the only second order Butterworth filter design and magnitude response which shows something like a bandpass filter. And in the bottom you will see which in this case is going to be an infinitely long impulse response of course.

1.33 Slide 38

This is the result when you increase the filter order from 2 to 10 and you see less smooth impulse response but then a better bandpass filter appearing.

1.34 Slide 39

Further increase the filter order from 10 to 18 again you see a sharper bandpass frequency response appearing.

1.35 Slide 40

Finally this is the result for 26^{th} order Butterworth filter. So again you see a sharper bandpass frequency response. This can be compared to FIR filter design so you if can have 26^{th} order IIR filter it has about 50 or so filter coefficients so you can compare that roughly saying runtime complexity with the 50^{th} order bandpass FIR filter design then you can compare whether one is more frequency selective than the other.

2 Chapter 5

2.1 Slide 2

In previous chapter, we have seen how a complete filter design process start, and the first step with defining the filter specifications. We start from, let’s say, an ideal passband, or a stopband filter response with particular characteristics of constraints. And then in the second step, you derive a transfer function, for instance, ultimately approximates that ideal response. And there we have seen FIR and IIR filter design to do that. In this chapter, we are going to consider filter realization. And realization is a step where you design a block scheme of a signal flow graph by putting together various building blocks. In the building blocks, there are going to be delay elements, multipliers and adders. So the idea is that signal flow graph that beats(?) together really realizes the transfer function as you have designed it in the second step. So the transfer function of the block scheme is indeed the designed transfer function. In the next chapter, and in the final step, we are going to consider filter implementation where you turn the realization into a true hardware or software implementation. And this is where finite word length issues are going to appear, which introduce or may introduce particular problems. So that in the end the implemented filter may indeed not behave as the desired sort of designed filter and then you would have to go back to previous steps, redo the filter design or redo the filter realization, consider other filter realizations or other filter implementation aspects.

2.2 Slide 3

So first we consider FIR filter realizations, so the realization of the finite impulse response filters. Again, filter realization is where you construct or realize linear time invariant system or a block scheme or a signal flow graph by putting together basic building blocks, and the basic building blocks here are delay elements, adders, multipliers, such that the input output behavior of the resulting signal flow graph or block schemes is given by the FIR filter transfer functions, or equivalently by the difference equations that specify the FIR filters as it is spelled out here in the formula. So the difference equations say y[k], the output of your FIR filter at time k, is a linear combination of the past so many input samples. So it is b0 times u[k], the input sample at time k plus b1 times u[k-1], so b1 times the previous input sample plus , etc., up to the bL times u[k-L], so make a linear combination of the last so many input samples. This is what your FIR filter really does.

In the next few slides, we are going to see several different possible realizations. We are going to see , sorts of, straight-forward realizations which refer to as the direct form realization, and the transposed direct form realization. And then, we are also going to two nontrivial realizations where the derivation of these realizations typically takes one page of mathematics. So that will bring us to lattice realizations, which are also referred to as LPC lattice realizations. And lossless lattice realizations. In the later part of the course, related to sub band systems and filter banks. We are also going to revisit FIR filter realization and develop what is referred to as frequency domain realizations. So that is the part that really belongs in this chapter. But because it relies on filter bank theory, we are going to have it on the further open of the course.

2.3 Slide 4

So first, direct form realization, this is really simple. It is basically the difference equation, which is immediately transformed or translated into block schemes with delay element and multipliers and adders. So the block of a single flow graph that you have here, you see the input signal at time k is feed into a delay line, and a delay line obviously produces, at the same point in time, produces the previous input samples u[k-1], u[k-2], up to, in this example u[k-4]. The multipliers and the adders basically make the linear combination b0 times the last input sample, plus b1 times the previous input sample etc. And you add everything together to produce the output signal. So this is an immediate sort of translation into a block scheme of the difference equation as you have it in the formula. This is really simple and it is referred to as the direct form realization.

2.4 Slide 5

To derive a second realization which is also pretty trivial and which is referred as the transposed direct from realization. We are going to start from the direct from realization of the previous slides and then apply a particular operation which is referred to as retiming.

Retiming is basically where you select a subgraph from the original graph. So if you consider the signal flow graph, and you select a subgraph which is the red shaded part. It is a subgraph and it has an input signal and it also produces an output signal. So that in itself, it is like a filtering operation. And if you consider that subsystem, the input signal is delayed by one delay element and then you could ask what happens really if I remove the delay elements from the input branch into the subgraph. If you do that, then basically, the internal operation of the red shaded part continues to be the same. So basically it is the same filter or the same subsystem producing a similar output signal from a similar input signal. The only difference is that the delay elements on the input branch, on the input signal, I should say, has been removed. So everything is passed onto the subgraph, one time sample or one sampling period earlier. The end result is that the output signal from the red shaded part is exactly the same output signal. Only it appears one sampling period earlier. So if you

did not add a delay on the output branch of the red shaded part, the end result is exactly the same. So reconsider the red shaded part has an input branch with a delay element. If you remove the delay element on the input branch and you move it onto the output branch, then the overall system has the same input output behavior. So this is a valid operation, and the operation is referred to as retiming. So the end result of the retiming operation is that one delay element in the top branch is moved into the bottom branch. And of course, this is an operation which can be repeated for each and every delay elements in the original graph. The end result is then indicated in the next slide.

So if you repeat this retiming operation to move every delay elements in the top branch into the lower branch, the end result is what you see in this slide. And this is referred to as the transposed direct form realization. Which should be realized here is that we have only applied operations that do not change the input output behavior. So this realization, if you compare it to the direct form realization if you feed it with the same input signal, it is going to produce the same output signal. But at each point in time, you are going to see different internal input signal into the realization. So if you would turn this into a piece of software of a piece of hardware, for instance with hardware multipliers and then adders and hardware memory cells for the delay operations. Then you are going to something which is different from what you would obtain based on the direct form realization but still with the same input output behavior. This transposed direct form realization could have specific advantages over the direct form realization. For instance, in a hardware realization, all the computations that you see sitting in between two delay elements basically form a computational path which is all the computations or operations, multipliers for multiplications, and addition that you have to do within one sampling period. And in the direct form realization, the longest computational path would be basically if you go back a few slides One multiplication and then the whole sequence of four, in this case, additions, whereas in this case, because of the delay elements sitting in between the additions, the longest computational path is basically only one multiplication plus one addition and that could lead to more efficient and faster hardware realization or implementation.

2.5 Slide 6

The next realization we are considering is the lattice realization, and it is a nontrivial realization in that as you are going to see, it will take one page of mathematics for the derivation. So we start from the same transfer function H(z) of the FIR filters which is given here as a difference equation. y[k], the output sample at time k, is linear combination of the past so many input sample b0 times u[k] and etc.

Now, something that we are going to be seeing in future derivations is that we are going to define a second transfer function, which is denoted here H tilde of z, which is a transfer function, where in a sense, we do not have any particular interest in that transfer function, but it will turn out that by co-realizing H[z] with the H tilde of z. We are going to realizing interesting structure. And the H tilde of z is defined here based on the same filter coefficients. And the first difference equation you have b0 up to bL from left to right. And the second difference equation which defines the H tilde, I have the same filter coefficients, b0 to bL, but now b0 sits at the right hand side and bL sits at the left hand side. We will have the same filter coefficients, but in reversed order. And that defines the H tilde of z, which we are going to be realizing together with the H(z). It turns out that by reversing the order of the filter coefficients, the H tilde of z is going to have the same magnitude response as the original H(z). So if H(z), for instance, is a designed LP filter, and then H tilde of z is also a LP filter. Only the phase response of H tilde of z is going to be different from the phase response of H(z). The last formula, which gives a simple proof for an indication of the statement that magnitude response for H tilde of z is the same as the magnitude response for the H. So if compute the magnitude response, you take the transfer function multiplied by the complex conjugate and evaluate that on the unit circle. Now if the transfer function has only real valued coefficients, then basically to obtain the complex conjugate, you replace z by z-1, and evaluate everything on the unit circle where you replace z by ejw, and if you then plug in the difference equation or the transfer function versions of the difference equation, you immediately see that indeed you end up with the same magnitude response.

2.6 Slide 7

So the lattice realization is going to be derived from an initial direct form of realization of the two transfer functions with H and H tilde. So what you have here as a starting point is the direct form realization where you see the input signal u[k] which is again fed into the delay line, and then you have basically two lines of multiplication with the b coefficients, and two lines of additions producing the y tilde output together with the y output. The sequence of the multiplications and adders producing the y output has the filter coefficient b0, b1 b2, b3, b4 from left to right. That is the second line with b coefficients. And if you add all the multiplied input samples together, you ended up obtaining the output sample y[k]. The other line, with b coefficients, has the same b coefficients, but now from right to left, you see, b0, b1, b2, b3, b4. So that realizes H tilde transfer function. And if you add the multiplied input samples together, you end up with y tilde at time k output sample. So this is the starting point for the derivation of the lattice realization. And this is where we are going to do one page of mathematics to derive basically the lattice realization.

2.7 Slide 8

So here is the one page of mathematics, before I start, I should say, this derivation here is given for the 4^{th} order FIR filter. If you see filter coefficients b0, b1, up to b4. Of course, the generalization of this is straightforward, so you can also apply the same derivation for the 5^{th} or 10^{th} of 100^{th} or whatever order filters. So it is a general realization or derivation of a realization that is given here. So starting point is the formula at the top of the slides, which is basically the representative of the direct form realization as we had in the previous slide. So we have two output signals which are given here in z transforms Y tilde of z and Y(z), which are equal to the right hand side of the equation. And you should read the right hand side of the equation from right to left. So you start with the input signal and z transform gives us U(z), which is first multiplied with a column vector, reminds the transpose which sits here. So the components of this column vector 1, z-1 to the z-4, which is basically representative of the delay line as you have it in the direct form realization. So it generates the delayed version of the input signal. And then finally you have the 2 by 5 matrix in this case with the b coefficients. So you have the first line from left to right b4 to b0, so that is the reversed sequence of b coefficients to produce the Y tilde output. And then the second line b0 – b4, which is the original sequence of B coefficients to produce the Y output. So as I mentioned, this is really this equation is really representative of the direct form of realization. What we are going to do now is to factorize the matrix that you have in this equation into 2 matrices as you have it in the next formula. So you have a 2 by 5 matrix in the first formula which is factored into a 2 by 2 matrix with K of zero. And then again a 2 by 5 matrix with a b prime coefficients in the second formula. So we are going to define a parameter which is denoted here as K0 and it is defined as b4/b0. So it is the last filter coefficient b4 divided by the first filter coefficient b0. So this is the definition of the K0, and K0 is used in the first 2 by 2 matrix in the new formula. So there you have one K0 at the first line K0 and 1 at the second line. And the second matrix is again a 2 by 5 matrix with b prime coefficients and a zero to the left and a zero to the right. But for b prime coefficients and the definition of the b prime coefficients is given at the bottom of the formula.

Now, to see that this is a valid factorization, you can basically prove the thing by working your way back from the second formula to first formula. So if you plug in the definition for K0 and you plug in the definition for the b prime coefficients, you can straightforwardly prove that the product of the two matrices is indeed the original matrix with a b coefficients. That is simply enough.

This derivation, you should notice that in two cases, it is not going to work very well. The definition of K0 is before divided by b4/b0. That obviously excludes the case where b0 equals 0. Now in that case what you basically have is an FIR filter with the first coefficient b0 equal to 0. So in the z transform of that FIR filter. The term multiplying z to the zero is gone and the first determining the transfer function is b1 time z-1, so every term has at least z-1. So that means you can form the transfer function from the FIR transfer function, you can extract a z-1 factor and then you get a transfer function with a

nonzero b0. From there on, you can follow this derivation. So in case of b0 equal to zero, you can easily find a sort of a workaround extract a common z-1 transfer function from the original transfer function and then continue as you have it here.

The other case is where K0's absolute value is equal to 1, so if K0 is equal to ether 1 or -1, then also the factorization is not going to work. The reason is that if you go to the 2 by 2 matrix with the K0, if K0 is for instance 1, then you have one 1 in the first line and one 1 in the second line which is a rank deficient matrix and similar for the case K0 equal -1, you are going to see a rank deficient matrix. And because of the rank deficiency of that 2 by 2 matrix, then also the factorization does not work. So if you encounter K0 is equal to 1 in absolute value. Then this factorization does not work and you cannot do the regularization as we have it in the next slide. You just have to give up.

Now, a remarkable thing is that the factorization that you obtain here from the original matrix has symmetry property similar to the symmetry properties of the original matrix. So the original 2 by 5 matrix has the b coefficient b4 from left to right to b0 in the top row and then b4 to b0 from right to left in the bottom row. So the bottom row is the reversed version of the top row. You have the same symmetry and factored form in a 2 by 2 matrix. You have 1 K0 in the top row and K0 1 in the bottom row. And similarly in the 2 by 5 matrix you have 0 and then b3prime to b0prime in the top row from left to right and then in the bottom row you have from right to left exactly the same sequence. So that is something to remember.

Now, let us again focus on this resulting 2 by 5 matrix with the b prime coefficients, Look at the second row of this matrix b0prime to 0, and then the 0 multiplies to the z-4 in the transposed vector with the powers of z. So z-4 is multiplied by 0 so it basically drops out, which means that the highest order term in the transfer function is removed, so that actually corresponds to a transfer function, which is a third order transfer function with coefficients b0prime to b3prime rather than a 4^{th} order transfer function as you had it initially with the H and the H tilde. So there is an order reduction happening here, which is represented by the zero in the second row of this 2 by 5 matrix. In the top row of the 2 by 5 matrix, you have the same coefficients, but then the trailing zero has been reversed into a leading 0. So there you have a basically transfer function where the leading coefficient multiplying by the 1 in the vector with the powers of z, so multiplying a one or z to the zero. So the leading term is equal to zero and then you have b3prime multiplying z-1, etc.

Now, this is, in a similar fashion, as we had if earlier up, you can from this TF extract a common z-1 factor and end up with, again, an order reduced TF. This is the 2^{nd} equation in the large formula where the 2 by 2 matrix with a K0 is unchanged. And then the 2 by 5 matrix is reduced to a 2 by 4 matrix with b prime coefficients by removing the trailing 0 in the last row and also the leading 0 in the top row and shifting the coefficients in the top row one position to the left. In order to be able to do that. You have to extract as I indicated a common z-1 and a corresponding TF. And that is the boxed z-1 in the 2 by 2 matrix that sits in the middle.

So the end result is an interesting factorization of the original two transfer function, H and H tilde. If you read the resulting formula, from right to left, you see that the two outputs Y tilde and Y are equal to the U(z) at the right hand, so the right hand part of the equation which is multiplied by a column vector with powers of z, we do not need z-4 here. Because the order reduction which has happened here. So that corresponds to a delay line which is applied the input signals with one fewer delay operation. And then the delayed versions of the input signals are multiplied by the b prime coefficients. So the bottom row b0prime to b3prime and the top row with the same b prime coefficients in reversed order. So these are the third order filters with coefficients which are reversed version of each other. Then you have two filtered versions of the input signal. One version, where you filter with the b prime coefficients from right to left and the second output where you filter with a b prime coefficients from left to right. One of these output signals is then delayed, multiplied by the z-1 and the other one is not delayed, and then the result of that is multiplied with a 2 by 2 matrix which has K0. So this factorization of the original 4^{th} order TF, and H and H tilde are two 3^{rd} order TF with b prime coefficients and then one delayed operation and then some sort of crisscross combination with the K0. It is representative of a new realization which is indicated in the next slide.

2.8 Slide 9

So in this slide you have the graphical representation of the formula in the previous slide. What you have in this slide is in the red shaded part, you see the input signal u[k], which is Fed into a delay line with three delays. Now, instead of four delays in the original direct form realization. So this is where you have the order reduction and then the outputs from the delay line are multiplied by the b prime coefficients, you have one line of b prime coefficients, first line with b three prime b two prime b, one prime b zero prime from left to right, and then a second similar line with b prime coefficients in reversed order. So b zero prime, b one prime, b two prime b three prime from again left to right. And by applying these multiplications and then the summations, you produce two intermediate output signals. So the output signals from the red shaded parts. Then the next operation is, remember the delaying of one of these two output signals. This is where you see the delay elements and then what you should observe is that all together again you have four delay elements, three in the red shaded part and then one delay here at the output of the red shaded part. So it has four delay elements as you had it in the original direct form realization. So that's, that's some sort of order preservation overall which you have here. So red shaded part is the filtering with the b prime coefficient filters that produces two outputs. One output is delayed and then the two resulting outputs are crisscross combined by means of this K0 matrix or coefficient. So one output y[k] is going to be equal to the delayed output multiplied by K0 plus the non delayed output and vice versa. The y tilde output is going to be the delayed output from the red shaded graph plus K0 times the long delayed part and that produces the y[k] and y tilde k. Can you remember that for the same input sequence u at time k? You're still going to produce the same output sequences y[k], y tilde k.

Now, the interesting part is that what you see in the red shaded part is a direct form realization of two filters with b prime coefficients and one filter has b prime coefficients b zero b one b two b three in the bottom row, and then the other filter has the same coefficients but then in reversed order, so from right to left, b zero, b one, b two b three, and this is exactly the same thing as we started from. So we started from a joint realization. Remember the H and H tilde. To FIR transfer functions with the same set of filter coefficients, only in reversed order. And we started from a fourth order system. Now the red shaded part is exactly the same thing, but it's a third order filters. So there's an order reduction happening here. This is where we have extracted one order or one delay operation, as you have it to the left of the red shaded part. And that also means that the previous derivation mathematics from the previous slide and the resulting realization can be repeated or applied to the red shaded part in this slide. And so if we repeat the procedure over and over again, we end up with a particular realization, which is referred to as the lattice realization.

2.9 Slide 10

So this slide shows the resulting, as we refer to it, lattice realization. And it's basically a structure where you see different sections or blocks and each block or section is defined by a K(Kappa) parameter. So, the K0, as we had in the derivation two slides back, and then a similar K1, K2 and K3. And so each section is basically this crisscross combination, which is based on a K coefficient. And in between two such sections, you have delay operations, where you see of the two output signals from one such Kappa section. One output signal is delayed and the other output signal is not delayed. And also if you, if you repeat the order reduction process as you had it in the previous slides. You keep on doing the same thing. In the end, what remains is something which, as you would see it is going to have only the first coefficient, b0, which always survives each and every order extraction. So, at the top right of the single flow graph, you see that the input signal eventually is going to be first multiplied by the b0 coefficient and then you enter this lattice structure where you see a delay operation and then a crisscross combination with K0, and then again, delay operation, etc. So this is referred to as the lattice realization and it's an interesting structure. It's should repeat here that this is equivalent to the original direct form realization. So if you have a direct form realization, or transposed direct form realization, you feed it with an

input signal, you can produces an output signal y[k] here, exactly the same thing. So you feed it with the same input signal. It's going to produce the same output signal y[k], but you see a completely different realization. You don't see the b coefficients anymore. Except for the b0, you see the K coefficient K0 up to K3, so the b coefficients are gone, have been replaced by the K(Kappa) coefficients.

2.10 Slide 11

The latest realization in the previous slide is also referred to as the linear predictive coding lattice and the K coefficients are referred to as reflection coefficients. For some reason that will not be explained here. The remarkable thing is that whereas in the original direct form realization, you see the b coefficients. Here in the lattice realization you don't see b coefficients anymore, you see K coefficients. There is a procedure which is actually indicated in the slide with a mathematical derivation to compute the K from the from the b coefficients, remember that K0 first reflection coefficient was before divided by b zero, so the last b coefficient divided by the first b coefficient. From the K0, then you compute the b prime coefficients and from the b prime coefficients you're going to compute the next K, K1, which is then going to be the last b prime coefficient divided by the first b prime coefficient, etc, on your go. So there's an algorithm to compute reflection coefficients from b coefficients.

The remarkable thing here is that this algorithm, this procedure to compute reflection coefficients is actually also known in a completely different context, namely in control theory. There's something which is referred to as a ‘Schur-Cohn’ stability test which comes down to executing exactly the same algorithm. And the problem which is addressed then is whether for a given polynomial. So in this case, it's the b of z polynomial whether all of the zeros of this polynomial are stable, so lie inside the unit circle. This is a difficult question because computing the zeros of a high order polynomial is a difficult task. And so the remarkable ‘Schur-Cohn’ stability criterion gives a solution to this and it basically says that you do not have to compute explicitly the zeros of a polynomial, you only have to compute the reflection coefficients based on the procedure as we added a few slides back. A statement here is that all of the zeros of the polynomial are stable, so lying inside the unit circle. If and only if all the reflection coefficients are bounded by one, in absolute value. So that's a very strong statement and so from the b coefficients of a polynomial you straightforwardly compute the reflection coefficients with a simple test on the reflection coefficients. It decides whether the zeros of the polynomial are indeed stable. So this is the ‘Schur-Cohn’ stability criterion or the stability test. If you go back a few slides, the mathematical derivation, you see that the procedure actually indeed breaks down if one of the reflection coefficients is encountered to be one in absolute value. So this is where the procedure breaks down, and you basically have to select a different realization. There's nothing wrong with the filter. It's an FIR filter, in this case a zero, at least one zero which is outside the unit circle. But for an FIR system, zeros can indeed lie outside the unit circle. That's not a true issue. So the link with the ‘Schur-Cohn’ stability test is remarkable, but not very relevant in this case. Further up, we're going to see a second case where there's a link with the your ‘Schur-Cohn’ stability criterion, and then it is indeed very relevant. The latest realization that we have derived here in itself is not perhaps an overly interesting realization, but the way we have derived. It, sort of, gives a procedure which we're also going to be using in future derivations, so in a sense it sets the stage for similar derivations in future slides.

2.11 Slide 12

The next realization that we're going to be discussing is referred to as the lossless lattice realization. The deration is similar to the derivation of the LPC lattice realization in the previous slides. So again we start from a transfer function, the H(z) with the b coefficients b0 up to bL in the first formula. And again we're going to realize this together with a second transfer function which is referred to here as the H tilde of z with a set of coefficients which are referred to as the b tilde tilde coefficients. So b tilde tilde of zero up to b tilde tilde of L, the order of the filter is the same as the order of the H(z) filter

and the question is what kind of b tilde tilde coefficients are we going to be using here. The basic formula that that will define the b tilde tilde coefficients is the last formula in the slides here, which I would refer to as the magic formula. Given the H(z), you're going to find an H tilde tilde of z that satisfies this equation. So H(z) times H(z-1) plus H tilde tilde z times H tilde tilde z-1 has to be equal to one and that's at this point a bit strange formula. So the first question is, what is the meaning of the formula? What is the interpretation of this formula? We're going to see it in the next slide. And then the second question is whether given the H(z), so the b coefficients, there is indeed a simple procedure to compute the b tilde tilde coefficients based on this equation, and that's in the next next slide. And the answer is indeed yes, there's a simple procedure to compute the b tilde tilde coefficients from the b coefficients.

2.12 Slide 13

The slide gives the interpretation of the magic equation and we obtain this by evaluating the magic equation on the unit circle. So we're going to be replacing every z by an ejw. The first term is H(z) times H(z-1), if you assume that H has real valued coefficients, then you can easily see that this term evaluated on the unit circle is basically the squared magnitude response of H. And in the second term is squared magnitude response of H tilde tilde. And so the magic equation basically says that the sum of the squared magnitude responses of H and H tilde tilde is uniformly equal to one so is equal to one for each and every radio frequency. The graph at the right gives a very simple example where you see the squared magnitude response of H, the lowermost part of the figure and H here is some sort of low pass filter, passing the low frequencies and attenuating the high frequencies. And then because of the magic equation and the sum of the squared magnitude responses having to be equal to one, you see that H tilde tilde must be some sort of high pass filter that attenuates the low frequencies. And passes the high frequencies. So wherever H is large H tilde tilde has to be small and vice versa. So if one is a low pass filter, then the other is a high pass filter. These filters are then said to be complementary. And because we're discussing squared magnitude responses, we say they are power complementary. The meaning is also that if you feed these two filters, you view them as a one input two output system. You feed them with an input signal corresponding to, for instance, the indicated radial frequency, and if the input signal has a certain amount of power, then what you're going to see at the output is that some of the input power appears at one output, let's say the H output and then the rest of the input power has to appear at the other output at the h tilde tilde output. So all of the input power reappears at the outputs. Nothing of the input power is truly lost. All the power is preserved and hence the system is said to be power complementary and are also lossless. There's no power lost in the system. The statement also holds for a general broadband signal which is a sum of radial frequency components. Of course, if you have a general input signal, broadband input signal with a certain amount of power and some of the input power is going to appear at the H output. And the rest of the input power is going to appear at the H tilde tilde outputs. So this is a power complementary system, also referred to as a lossless system.

2.13 Slide 14

Next question that we have to answer is whether there is a simple procedure to compute the H tilde tilde, so b tilde tilde coefficients from a given transfer function H(z), so from a given set of b coefficients. And the answer is yes, there is a procedure and even though we're not really interested in the details of the procedure, the slide here shows what sort of the outline is. So, from the magic equation the unknowns are the H tilde tilde and so H tilde tilde z times H tilde tilde of z to the minus one is equal to one minus H(z) times H(z-1) and this is referred to as R(z). If you inspect the R(z) then it seems to have a special property. If R(z) has a root Zi as it is indicated here then 1/Zi also has to be a root of the R(z), so for every Zi you have a 1/z and so if you factorize the H tilde tilde of z times H tilde tilde of z to the minus one in factors corresponding to the individual roots of R(z), you get factors that belong to the Zi roots and factors that belong to 1/Zi roots. The main observation then is that this factorization you basically have to split it in two parts. A part that corresponds to the H tilde

tilde of z in the left hand part of the equation and a part that corresponds to the H tilde tilde of z to the minus one right. And because of the special structure of this factorization, it's just a matter of assigning roots to H tilde tilde of z and then assigning the corresponding 1/Zi roots to H tilde tilde of z to minus one, so that's a simple thing. Let’s give some degrees of flexibility of freedom. You could for instance pick all the roots that lie inside the unit circle and assign these to H tilde tilde. That leads to what is referred to as a minimum phase H tilde tilde FIR filter. This factorization is also referred to as spectral factorization. So I'm not giving details of the procedure here the main idea is really simple and the main message is if you give me an H(z), there is indeed a procedure to compute in H tilde tilde of z and this is all we need to know.

2.14 Slide 15

The starting point to derive the lossless lattice realization is the direct form realization of the two transfer functions H(z) and H tilde tilde of z as you have it here. So you see. Input signal u[k] feed into a delay line and then the delay line is tabbed into two lines. One line with multipliers with the b coefficients on one line with multipliers with the b tilde tilde the coefficients. And if you sum all the multiplied delayed signals, you end up with the y tilde tilde output and the y output. From here we're going to use again one page of mathematics to do some sort of order reduction similar to the order reduction that we had in the LPC lattice derivation. So the mathematics are going to be similar. The procedure is going to be similar, but the details of the mathematics are going to be different, of course.

2.15 Slide 16

So here's the one page of mathematics. Starting point is the formula at the top, which is again representative of the direct form realization, as we had it in the previous slide. So it produces two outputs, Y tilde tilde of z and Y(z). And the way these outputs are produced is spelled out in the right hand part of the equation. If you read it from right to left, you start with the input signal. You have u[z], which is fed into a delay line represented by the column vector with the delay operators 1, z-1, z-2 up to z-4, and then the outputs of the delay line are linearly combined by means of the b tilde tilde coefficients to produce the Y tilde tilde output and the b coefficients to produce the Y output. And put all the coefficients and the b coefficients for the 2 by 5 matrix in the equation. Again, this derivation here is given for fourth order systems or fourth order initial H(z) FIR transfer function. Hence, you have a two by five matrix in this equation. But of course, everything easily generalizes to other system or filter orders, so you can do the same derivation for a filter order which is equal to five or ten or 1000 whatever. So, this is the starting point and I would like to do some sort of order reduction as we have done it also in the derivation of the LPC, the lattice realization. The order reduction step here is based on a formula which is the second formula in the slide here and which basically says b0 times b4 plus b tilde tilde of 0 times b tilde tilde of 4 is equal to zero. That sounds like a strange formula, but the origin of this is basically, again of course, the magic equation, and it's spelled out in small fonts in the footnote at the bottom of the slide.

And the magic equation says H(z) times H(z-1) plus H tilde tilde (z) times H tilde tilde (z-1) is equal to one, the meaning of the one is that this is basically a power series in z and z-1 where only one term is not zero and that's the one, so the term in z to the zero has a coefficient equal to one and all of the other terms in z, z to the minus one, z two, z to the minus two, etcetera, have coefficients which should be equal to zero. Now, if you substitute the z transforms for H(z) and H tilde tilde of z, so with the b coefficients and the b tilde tilde coefficients. And you multiply everything together and you group terms that have the same power of z, then you obtain what you have at the in the right hand side of the equation. For instance, the highest order term in z to the four has a coefficient which is spelled out here and actually the z to the minus 4 is exactly the same coefficient. And you can group them together and the coefficient is indeed this b0 times b4 plus b tilde tilde of zero times b tilde tilde of four and this coefficient has to be equal to zero. And so this is the formula as we have it in

the middle of the slides. So that explains this starting point b0 times b4 plus b tilde tilde of zero times b tilde tilde of four is equal to zero.

Now, an interpretation of this formula is in terms of two vectors. So two vectors has two components. And this equation, something equal to zero, is basically an inner product of two vectors, and it's spelled out in the same formula. The two two vectors are as follows. The first vector has two components, b tilde tilde 0 and b zero. And the second two vectors has two components, b tilde tilde four and b4, and so the inner product of these two two vectors is equal to zero. That's what we had derived at the bottom of the slide. And whenever two two vectors, whenever two vectors have an inner product which is equal to zero, the meaning is that these vectors are orthogonal. So what we have here is two orthogonal two vectors, b0, b tilde tilde of zero, and b tilde tilde four, b4, so these two vectors are orthogonal. You should now go back to the first equation. It turns out that these two vectors are in fact the first column vector and the last column vector of the two by five matrix. So the conclusion is if you in the starting equation, the first column vector in the two by five matrix in the last column vector in a two by five matrix are two orthogonal vectors. Now, if two vectors are orthogonal, then you can find one rotation angle that rotates one of the vectors, for instance, onto the x axis so that its y component becomes equal to zero. Whereas the same rotation rotates the second factor onto the y axis, so that its x component, its first component, becomes equal to zero. And this is exploited in the last formula at the bottom of the slide here where the two by five matrix from the first formula is factorized into a product of a two by two matrix and a new two by five matrix. The two by two matrix is a rotation matrix. When you apply this to a two vector, it executes a rotation over a rotation angle theta of zero. So it has cosines and sins of the theta0 rotation angle. And so this is the rotation which you apply to the two two vectors such that one is rotated onto the x axis and the other one is rotated onto the y axis. So for one, the y component is going to become zero and for the other one, the x component is going to become zero. And this is what you observe in the new two by five matrix. You see two zeros appearing and a new set of b coefficients which are referred to as b prime coefficients. So b prime and b tilde tilde prime coefficients which you have the two zeros. There you have a zero in the top left corner and a zero in the bottom right corner. And it's because you've picked the right rotation that you have obtained these two 0 coefficients. The meaning of this is an order reduction. This is exactly the same thing as we had in the derivation of the LPC at lattice. So if you start from the zero in the bottom right position in the two by five matrix, it multiplies the z to the minus four. The column vector with a delay operators and so the z to the minus four term drops out which means that the transfer function is reduced from a fourth order transfer function into a third order transfer function. In the first row of the new two by five matrix, the zero sits at the left hand side in a two by five matrix top left you see a similar zero. Now it's at the beginning of the row, but that we know how to handle you extract from the resulting, from the corresponding transfer function, you extract a common z to the minus one, that's the shaded or framed part in the last equation, the z of the minus one that you see appearing there, and then all of the other. Be till until the prime coefficients are shifted one position to the left and similarly as you had for the last or for the second row in the two by five matrix, you indeed achieve in order reduction.

2.16 Slide 17

The slide gives a graphical representation of what we have derived in the previous slide. So the starting point was a system with one input u[k] and two outputs y tilde tilde of k and y[k], and it was given in direct form realization. And it was originally a fourth order system with four delay elements and so five b coefficients and five b tilde tilde coefficients. The result after the order reduction step of the previous slides is given here, which is a cascade of a number of operations. And the first operation is the red shaded part which is basically an order reduced direct form realization. So you see one and the same input signal u[k] and two output signals, intermediate output signals which are produced in this part. This is order reduced. So it's sort of three, you see three delay elements and so four b prime coefficients, b prime zero, b prime one, b prime two, b prime three, and similarly there b tilde tilde prime coefficients in the top row of multipliers. So this

is third order, it's sort of reduced and it produces two intermediate output signals. And then from the formula in the previous slide, you learn that one output again has to be delayed so it runs into a delayed operation and the other output is not delayed. And then the delayed outputs together with the not delayed output are fed into the rotation operation. So this is where you see the multiplications with the cosine and the sine of the rotation angle theta0. Ultimately producing the same output signals y[k] and y tilde tilde of k, so if we have achieved an order reduction, the red shaded part is third order where the starting point was a fourth order. And then the question arises whether we can repeat this order reduction step if we can again extract the first order section with a orthogonal rotation and reduce the direct form realization from third order to second order and then to first order, etcetera. The basic question whether we can indeed repeat this order reduction step is whether the red shaded parts and the corresponding transfer functions satisfy a magic equation because it was a magic equation that was the starting point for the derivation. So if we want to redo the derivation, we have to have to transfer functions that satisfy this magic equation. It basically means that. The red shaded system with one input u[k] and the two intermediate output is again a lossless system or a power complementary system. Now this is something that can be proved and proof is given at the bottom of the slides and you have to look into the details. You can also use your intuition in a sense if you observe that. The original system was power complementary lossless so it means all the power that is fed to the system at the input at the u[k] reappears at the outputs at y tilde tilde of k and y[k]. Now the last part of the realization here is the orthogonal rotation with the rotation angle theta0. You remember from your linear algebra courses that orthogonal rotation preserves the norm of a vector, and the norm is the sum of the squares of the component. So it's basically the energy, and in a two vector, which is preserved by an orthogonal rotation and energy divided by time is power. So basically the orthogonal rotation with the rotation angle theta0 preserves the power of the two input signals, so in other words, the sum power of the two input signals to rotation is equal to the sum power of the output signals from the rotation, and of course the delay operation also doesn't consume any power. So then everything that you see in the left hand part of the graph here is lossless preserves. Power in the input signals so the power in the input signals is preserved and reappears in the output signals, so the overall system is known to be lossless from input u[k] to output y tilde tilde and y. And then also, the last part is known to be lossless. Then you can sort of work your way back and decide that from the input to the intermediate outputs from the red shaded parts. That system in the direct form realization must also be a lossless or power complementary system, hence satisfy a magic equation. So this is the intuition behind the proof that is given at the bottom of the slides. And it basically tells you that you can repeat the order reduction process. So now you have a third order system in the red shaded parts. You can then reduce it to second order to first order to zero order.

2.17 Slide 18

If you repeat the order reduction step from the previous slides over and over again, you end up with what it's referred to as the lossless lattice realization. It's a lattice realization. It looks very much like the LPC lattice realization that we have derived earlier up. But now you see that each section has an orthogonal rotation with a rotation angle theta. So each section in this lattice is a two input two output system that takes two intermediate input signals to produce two intermediate output signals. And the output signals are formed from the input signals by applying an orthogonal rotation. So you see, and from the original fourth order system we have derived a fourth order lossless lattice realization here, rather than the original b coefficients to b0, up to b4, what you see here is rotation angles from theta0, as you had in the previous slide up to theta4 at the far right of the slide. So this is the lossless lattice realization.

2.18 Slide 19

The lossless lattice realization is sometimes also referred to as the paraunitary lattice realization per paraunitary as something we're going to be studying later in the course in the part of subbank systems and filter banks. So at this point you can safely ignore the term. So this is again a realization, an alternative realization for the original trivial, let's say direct form realization. It looks very differently, but it produces the same output signal y[k] when it's fet with the same input signal u[k] compared to the original direct form realization. The remarkable thing about the lossless lattice realization is that it consists of lattice section and each lattice section is a two input two output section that takes two intermediate input signals and from there by applying an orthogonal rotation produces two intermediate output signals, as was also indicated in the previous slide, as it was also indicated in the previous previous slide. An orthogonal rotation preserves the norm of the two vectors. So if you apply an orthogonal rotation to two vectors with two components IN1, IN2. As it is denoted here, the output is two output components, OUT1, OUT2, then the sum of the IN squares is equal to sum of the squares of the output components. So altogether the energy, you could say, of the input vector, is preserved. So the energy of the output vectors is equal to the energy of the input vectors. And if you divide by time, then you end up with power. So basically it says if you take two input signals and the sum power of the two input signals is equal to the sum power of the two output signals, so this is where each section in the lossless lattice realization each orthogonal rotation is a lossless system. And in between the sections, you also have delay operations, but obviously delay operations are also lossless operation. So from the lossless lattice realization, as you only see basically orthogonal rotations together with delay elements, you can immediately decide that this has to be a lossless system and it's in a sense the time domain view on losslessness which is equivalent to the earlier frequency domain view on losslessness as we had basically as our starting point we started. Remember? From an initial transfer function H(z), and we've complemented it with a second transfer function H tilde tilde of z and the two together have to satisfy the magic equation where the interpretation of the magic equation was indeed that the two transfer functions H and H tilde tilde are power complementary or lossless, right? So we've started from two transfer functions which are together lossless and we've turned them into a realization where you can see the losslessness in the realization immediately, because that realization has only orthogonal rotations.

2.19 Slide 20

In the previous slides we have derived a lossless lattice realization, which was basically a system with one input signal producing two output signals. It turns out you can easily generalize this into lossless lattice realization that has one input signal and then more than two output signals in general N output signals. The example here has three output signals produced from one and the same input signal. The revision relies again on the transfer functions corresponding to all of the output signals satisfying some magic equations. So in this case you have three outputs. So three transfer functions H(z), H tilde (z)and H tilde tilde(z) and altogether they have to satisfy magic equation which is given here at the bottom of the slide. So, H(z) times H(z-1) plus a similar term for H tilde (z) plus a similar term for H tilde tilde (z) has to be equal to one and the meaning of this magic equation is again that all these three transfer functions together form a lossless system. So a power complementary system which means that if you feed the system with an input signal and the input signal has an amount of power, then this power reappears in the output. So some of the input power appears in the y[k] outputs some of the power appears in the y tilde[k] output and the rest of the input power appears has to appear in the y tilde tilde[k] output. So, you start from three transfer functions. In this case H(z), H tilde H, tilde tilde. You can do a similar derivation to do an order reduction and each order reduction in this case is going to extract two orthogonal transformations within this case two rotation angles as you have it here in the lattice section of which the details are given with a rotation angle theta1 and theta2. In general, so, in an n outputs lossless lattice system, each lattice section is going to have n-1 rotations and so rotation angles. It's actually a challenge to do the derivation here starting from the transfer functions. It's something you should try for for sure to see how the rotation angles are defined from the initial coefficients

of the transfer function. So, the end result is a lossless lattice realization. As you have it here, each lattice section has in this case, two orthogonal rotations. In the general case, it has n-1 orthogonal rotations. Also, notice the details of how the rotations are applied to the signals. So each lattice section takes three input signals produces three output signals. First, rotation rotates the first and the second input signal producing a first output and an intermediate second signal and then the intermediate second signal is rotated together with a third input signal to produce the second and third output signal from the lattice section. Again in between two lattice section you see intermediate signals flowing out of one lattice section and into the next lattice section and then one of the intermediate signals is delayed and the other signals are not delayed. So in the end you're going to see as many delay elements as you have lattice sections roughly which is the order of the system, assuming we start from transfer functions H, H tilde, H tilde tilde, all have the same filter order so this is the generalization to one input and output. system lossless lattice realization. This also ends the part on FIR realizations. And now the question is can we do similar realizations for IIR systems and this is going to be discussed in the second part of the lecture.

2.20 Slide 36

The second part of the lecture deals with IIR filters and IIR transfer function. So an IIR transfer function is a rational transfer function has a B(z) polynomial divided by an A(z) polynomial. So the B(z) has the b coefficients b0 up to bL for an L order system and A polynomial has a coefficients where a0 is normalized to one has always and then a1 up to aL for an L order system. Question is how can you now realize a rational transfer function and a IIR transfer function of this kind. Can you do similar trivial realizations like direct form, transposed direct form and that will turn out to be indeed easy? And then can you do the non trivial realizations, the lattice realizations. Here again, we're going to be discussing two lattice realizations. One is referred to as the lattice ladder realization. The other will again be referred to as lossless lattice realization from the direct form and transposed direct form realizations if an IIR system are also going to briefly look into parallel and cascade realizations, which are oftentimes used in practice.

2.21 Slide 37

This slide shows the first realization for an IIR system, and it's referred to as the direct form realization. If you start from the rational transfer function H(Z), B(z) divided by A(z), you can basically dissect it into two parts. First part is B(z), of course, and then this is in a sense multiplied by a second part, which is 1/ A(z). To drive a realization, you can start by realizing the two parts separately. And then somehow merge the two parts. And this is done in this slide. In fact, leading to the direct form realization, the top part of the flow graph realizes 1/A(z), the 1/A(z) part and the bottom part of the single flow graph realizes the B(z) part. The bottom part, you immediately recognize a direct form realization of the B(z) part, which is indeed an F I R filter. So you see a delay line which is Fed by an input signal to the left and then you tap the delay line. You multiply with the b coefficients. Some have everything together to obtain an output signal. The top part is similar, but it's a feedback system. So you again have a delay line and delay line is where the outputs from the delay line are multiplied by the a coefficients and these are feedback and then added to the input signal. And the resulting signal is then fed into the delay line. So here you can easily check that the top part in itself indeed realizes the 1/A(z) and the merger of the two parts, the one over A(z) parts with the B(z) parts basically comes down to reusing the same delay line, both for the top part and for the bottom part. So here you have your direct form realization. You can easily check that the transfer function from input to output for the system is indeed the rational function with the B(z) and the A(z) polynomial. It's interesting in every IIR realization. It's interesting to see what happens in a special case when all of the a coefficients are zero because this is the case where the H(z) reduces to only the B(z) part, right? And then you have an FIR system. So you should recognize an IIR realization indeed here when you put all the a coefficients in the top part equal to

zero. There's no feedback part anymore and the only thing that remains is the bottom part, which is indeed a direct form realization of the B(z), so this direct form IIR realization generalizes a direct form FIR realization.

2.22 Slide 38

Similar to the transposed direct form realization for the FIR filters, you can have a transposed direct form realization for IIR systems. The derivation of this is something we're not going to give the details, but it basically has a similar starting point. You have the two parts, the B(z) part and the 1/A(z) parts. In the previous slide, you first realized 1/A(z at the top of the slides and then B(z) at the bottom of the slides. Now you do it the other way around. So you first realized B(z) and then realize 1/A(z) and if you then somehow try to merge these two realizations and also as you headed into derivation of the transposed direct form for FIR systems to a retiming operation. Basically to move delay elements into the central line or branches in the signal photograph here. Then you end up with this particular transposed direct form realization. So the derivation is simple but the details are skipped here. The end result is this transposed direct form realization. Again, you can check what happens in a special case when all of the a coefficients are zero, then the bottom part basically drops out. So only the top part remains and this leads the transpose direct form realization of B(z), so this transpose direct form IIR system indeed or IIR realization indeed generalizes the transpose direct form realization for FIR system.

2.23 Slide 39

As we're going to see it in one of the later chapters, if you do direct form, transpose direct form realizations of IIR systems and in particular high order IIR systems. So if you're realizing high order rational transfer functions H(z), then doing a direct form realization or transposed direct form realization is actually a bad idea. In finite word length implementations, it could lead to very unpredictable results. And this is something that we're going to be studying actually in chapter seven, so high order systems for high order systems, direct form and transposed direct realizations are a bad idea. What is used as an alternative in practice then is something which is referred to as a parallel realization or a cascade realization. The principle is really that you start from your high order rational transfer function H(z)and you split it into lower order parts. There's basically two ways of splitting a high order rational transfer function into low order parts. The parts can either be a sum of terms and that leads to a parallel realization or you can either have a product of factors and that leads to a cascade realization. So first a parallel realization and that is based on partial fraction decomposition. So in your mathematics courses you have learned how you can split rational functions into some lower order rational functions and the formulas given here where we're not going to go into the details but it's a sum of terms and to realize a sum of terms you basically have to realize the individual terms individually and then sum the output. So that leads to the parallel realization as you have it in the right hand part of the of the slides, the graphical representation. So each term each lower order term is realized individually because it's a lower order term you may want to use direct form realizations there and that leads to parallel realization. There could be specific finite word length implementation issues with parallel realizations and in the realization for instance, of transmission zeros. But this is something that we're not going to be discussing here. The other alternative is to split your high order rational transfer function H(z) into a product of factors which is based on pole zero factorization. So basically you factorize your B(z) polynomial into a product of factors based on the zeros of the B(z) polynomial. And similarly for the A(z) polynomial, you split it or you factor it into a product of factor based on the zeros of the A(z) polynomial and then emerge basically two zeros of the B(z) with two zeros of the A(z) into a rational second order rational transfer function. And in the end, you're going to see a product of second order rational transfer functions. Typically, such second order sections are then referred to as biquads as you have a product of rational transfer functions basically the product of transfer functions through realization is going to be a cascade realization where you cascade all the individual second order realizations. Again, because it's only second order realizations you can use direct form. So

this is lower order rational transfer functions. So then it could be okay to use direct form realizations and that is going to lead to a cascade realization. As it is based on picking zeros from the B(z) polynomial and merging these with zeros of the A(z) binomial, so poles of your system, there are obviously multiple ways of pairing poles with zeros and defining then the second order sections. So this cascade realization is not unique and there's somehow heuristic rules as to how you should pair poles and zeros to have the best possible realization. So these are parallel and cascade realizations and these are oftentimes used in practice as a good alternative for direct form, transposed direct form realizations of high order rational transfer functions.

2.24 Slide 40

So now we're going to look into the non trivial realizations for IIR systems, the lattice realizations, we will have two of them. First in the slide we have the lattice ladder realization and then we're going to drive a lossless lattice realization which is similar to the lossless lattice realization for FIR systems. So first the lattice ladder realization we start from the H(z) transfer function with the numerator polynomial B(z) with the b coefficients divided by the denominator polynomial A(z) with a coefficients. Then whenever we do a lattice realization, we combine this initial transfer function that we would like to realize with a second transfer function, as I've mentioned earlier, you don't really have a particular interest in this second transfer function, but by joining the two transfer functions together, it always seems that an interesting realization is going to appear. So here we define a second transfer function. It's denoted as H tilde of z and has again a numerator and a denominator polynomial and these are defined as follows. If you consider the denominator polynomial, first A(z) is exactly the same denominator polynomial as you had it in the original IIR transfer function H(Z), so H tilde have the same denominator polynomial. Then the numerator polynomial of H tilde of z is denoted as A tilde of z and what you see there is that the coefficients of the A tilde polynomial are the same as the coefficients in the A polynomial. We use them in reversed order. So in the A polynomial you have coefficients from left to right. First one is the normalized coefficient equal to one and then you have a1, a2 up two a L, whereas in A tilde you have the same set of coefficients but now from right to left. So it starts with normalized coefficient equal to one and then a2 and then up to aL at the left. So then you can ask what is the meaning of this H tilde transfer function? We have seen earlier that if you take a polynomial and then you reverse the order of the coefficients, that this basically preserves the squared magnitude response of a polynomial. So if you check this squared magnitude response of the H tilde. As you have it in the formula at the bottom, right, so the squared magnitude response of H tilde. It is basically the squared magnitude response of A tilde divided by the squared magnitude response of A, and both are the same. So you should divide a thing by the same thing. Then you end up with something which is uniformly equal to one, a different way of presenting this is by seeing that H tilde also satisfies in itself something like the magic equation that we had in the case of the lossless lattice FIR derivation. So H tilted of z times H tilted of z to the minus one is equal to one, you can easily check that this is indeed a true statement if you substitute the expression for H tilde. And that means that H tilde in itself satisfies a magic equation. And a magic equation refers to losslessness. So H tilde in itself is a lossless filter. And now this is a single input single output filter. So it's a single input single output lossless filter, which is also referred to as an all pass filter that passes all radial frequencies, gives all radial frequencies magnitude response equal to one and that is in fact what we had derived in the left hand part of the equation. So this is how we define the H tilde of z, the second transfer function that we're going to be realizing together with the H(z) transfer function.

2.25 Slide 41

Starting point is again a direct form realization of two transfer functions from the previous slides. So the H(z) together with the H tilde of z. So what we have here is direct form realization with the feedback part that sits at the top of the graph

with a coefficients a1, a2, a3, a4. Again, this is a fourth order example, but we can say that everything easily generalizes to higher order systems. So you have the feedback parts with the a coefficients, which is actually shared by the two systems or filters. And then the bottom part is the multiplications with, on the one hand, the b coefficients to provide the y[k] output. And then rather than the multiplication with the b coefficients, the multiplication with a coefficients again, but now in reversed order to produce the y tilde output. So notice that in the feedback part you have the a coefficients at the top from left to right, a one, a two, a three, a four whereas. The bottom parts and the feed forward part of the H tilde filter, you see the same coefficients, but now from right to left, a one, a two, a three, a four, and you also see the leading normalized equal to one coefficient a0 is equal to one. So now comes one page of mathematics again to extract a first order section from this direct form realization.

2.26 Slide 43

The result from the order reduction step in the previous slides is shown graphically in this slide. So you see a signal flow graph of a system that still has the same same input signal u[k] and produces the same output signal slide to y tilde[k] and y[k], the signal flow graph basically has two parts. To the left you see the first order section that has been extracted. And to the right, you see a direct form realization of two third order filters which is basically third order, it's one order less than the original fourth order system that we started from. In between the two parts, so we're in the middle. You see signals flowing from left to right and from right to left at the top, you see an intermediate signal which is fed from the left into the right hand part, which is the input signal to the direct form realization part of the graph. And then the direct form realization part of the graph produces two intermediate output signals, which are propagated from right to left in the middle at the bottom of the of the graph. This is where you see that one of the intermediate output signals is delayed. This is the delay element, so we're in the middle of the graph. The other intermediate output signal is not delayed, so the left hand part is the extracted first order section. You see at the top rotation an orthogonal rotation with a rotation angle theta0. In the way the theta0 is computed when computed was indicated in the previous slide, you see a rotation that takes two input signals and produces two output signals. Notice that the input signals come in from the top left in the bottom right. And the output signals are now produced at the top right and the bottom left. So this is an orthogonal rotation. And next to the orthogonal rotation in the first order section, you also see one additional tap with a multiplication with one coefficient. And in this case it's b4 coefficient. So this is the extracted first order parts. To the right, you see the direct form realization of two third order transfer functions should produce the intermediate output signal, which would then propagate it into the first order section. And if you inspect the coefficients of this third order system, you see expressions which are basically functions of the a coefficients and then also have the b coefficients. The interesting part is that the coefficients, the expressions for the coefficients in the feedback parts, if you look into these formulas. You're going to see exactly the same formula in the feed forward part, somewhere in the middle of the graph to feed forward part that produces the intermediate y tilde output signal. So the row in the middle with the multipliers if you read the multipliers from right to left, you see the leading one, which is always, the normalized coefficient. Then you have a coefficient which is equal to a1 minus a4 times a3 divided by one minus a4 times a4, exactly the same expression appears in the feedback part if you read from left to right. So there also you see the same expression. So altogether the coefficients that you see in the feedback part or the coefficients that you see in the feed forward part producing the y tilde intermediate output signal. Again the order is reversed. So the third order system that you have to the right of the slide here is as exactly the same structure as the structure of the original fourth order system that we started from. Producing two output signals where the two output signals or transfer functions share the same feedback parts. And one of them has a specific feed forward part with the same coefficients in reversed order. So based on this observation, you see that you have a similar structure as the one you started from. So you can basically repeat the order reduction step. As you had it in the previous slides. So we're going to repeat the order reduction step over and over again until we only have first order sections since you have it in the left hand part of this slide.

2.27 Slide 44

When you repeat this order reduction step, as we had in the previous slides you end up with realization, which is shown graphically here in this slide, it's an alternative for the original direct form realization. So if you feed it with the same input signal, u[k], it's going to produce the same output signals still the y tilde [k] in the y[k]. This is referred to as the lattice ladder realization. It has all parts, which is basically the lattice parts which has orthogonal rotations similar to the orthogonal rotations we have seen in the lossless lattice FIR filter realizations. And then it has the ladder parts, which is the bottom parts, which indeed looks a bit like a ladder, you could say. So each section has an orthogonal rotation, and then a ladder part, orthogonal rotation has a rotation angle theta0 in the first section, and then theta1, theta2, theta3, so altogether four rotation angles for a fourth order system. And if you zoom in on the top part of the lattice part, which you should also notice is that you have signals flowing from left to right in the top line. And then these are a sense at the far right of the lattice section, bounced back into the lattice structure and then propagate from right to left in the lower branch. So each orthogonal rotation takes two input signals. One input flows in at the top left and one comes in at the bottom right. And then you combine these with the cosines and the sines of the rotation angle, producing two output. One output appears at the top right and one output appears at the bottom left, so you have again signal signals flowing from left to right in the top branch and then it bounces back at the far right side. The signal flowing from right to left in the lower branch. In the lower branch, this is also where you have delay elements in between every two successive lattice sections. So this is the top parts and then the lower part, the bottom parts is the ladder parts where you tap signals from the lattice parts, multiplying with the b coefficients, the b coefficients are b reading from left to right as before in then b3 prime. And then b2prime prime, b1prime prime prime. There's a particular way of computing these coefficients, of course, has developed in a mathematical derivation. And so you tap signals from the lattice part to a linear combination with the b coefficients to produce eventually the output signal y[k]. So this is referred to as the lattice ladder realization.

What we have in this slide is a slightly different version of exactly the same thing. It's a different realization also referred to as a lattice ladder realization. So realization of the same transfer functions, if you feed the realization with the same input signal u[k] still produces the same output signals y tilde [k] and y[k], it's the same thing. You still see the lattice part and the ladder part but the differences in the details. If you zoom in onto the lattice part, you see orthogonal rotations with the same rotation angles, theta0, theta1, theta2, theta3, but now the delay elements have been moved from the bottom line of the lattice section into the top line. In the top line you have signals from flowing from left to right and then this bounces at the right hand side of the slide into the lower branch, but now the delay elements have been moved from the bottom branch into the top branch. The ladder part also takes a different set of intermediate signals, so now it's branches from the top line also instead of from the bottom line of the lattice part, and in the ladder part you're also going to see a different set of b coefficients now. So now it starts with a b0 at the left and then a modified b1prime. And then b two prime prime and b three prime prime. And there's a procedure, of course, to compute these b coefficients from the initial set of b coefficients. So this is a slightly different version of the same thing. And it will turn out that the interpretation here is a bit nicer in a sense. You can already see that in a special case where all of the rotation angles are equal to zero, then cosine of a rotation angle is one, and sine of a rotation angle is equal to zero. If all the cosines are equal to one and all the sines are equal to zero. And basically, in the top line, you have signal propagation from left to right, and the signal is not changed in the rotation cells, it only flows from left to right and you have delay elements in between the rotation cells and in the in the bottom line again. The bottom line of the lattice part because the rotations are, in a sense avoided. There's no signal change. It's basically the signal flowing from left to right in the top line with the delay elements, and then it bounces back. At the right hand side of that slide and it's propagated unchanged to the y tilde output. So that basically says that the whole lattice part is replaced by only a delay line with delays in the, in the top Line. And that leads to a simple interpretation, as we're going to see it in one of the next slides.

2.28 Slide 45

There are few things that can be said about this derivation and the lattice ladder realizations that we had in the previous two slides. First, if you go back to the mathematical derivation, I've indicated that the way the rotation angles are computed which you see appearing. There is a procedure which we have come across earlier which has been referred to as a 'Schur-Cohn' stability tests or algorithm to compute reflection coefficients from a set of coefficients of a polynomial. So here the way that actually the sines of the rotation angles, the sines of the theta are computed. These are equal to reflection coefficients computed from the coefficients of the A polynomial. And this all makes a lot of sense because we started from a rational transfer function H(z) with a B(z) divided by an A(z) polynomial and this is supposed to be a stable system which means that the pulse of the system lying inside the unit circle, which means that the zeros of the a polynomial indeed lie inside the unit circle. If this is the case. So if the A(z) polynomial has its stable zeros, then the 'Schur-Cohn' stability criterion guarantees that if you compute reflection coefficients the K from the a coefficients. Then these needs are going to be bounded by one in absolute value, and that makes a lot of sense because the reflection coefficients are then set equal to the sines of the rotation angles and sines have to be bounded by one as well. In absolute value, you can check for yourself what happens when one of the rotation angles or one of the reflection coefficients is equal to one in absolute value. That's a special case. And it's interesting to, sort of, find out what happens in the derivation there. But altogether we see that the 'Schur-Cohn' stability tests as we have it seen appearing in the derivation of the LPC lattice for the FIR. In the FIR case and there it didn't have any particular meaning. Because stability of FIR systems is guaranteed. Loss of a B(z) polynomial can safely lie outside the unit circle. That's not an issue there, but here again remarkably we see this 'Schur-Cohn' stability criterion appearing and in this case it does make a lot of sense. The other thing we can notice is that in the realization if you zoom in onto the lattice part, that's the bottom part of the slide here. The lattice part has the orthogonal rotations and as it has been indicated in earlier slides, in orthogonal rotation takes a two vector. Rotates this into another two vector and it preserves the norm of the two vector and norm means energy and divided by time it means power. So it preserves power of two signals. So, that some power of two input signals to an orthogonal rotation is equal to the sum power of the output signals. So, each orthogonal rotation is a lossless operation. And then if you check the entire lattice part of the realization it consists of rotations which are lossless and then also delay elements which are also lossless. So altogether this, this lattice part of the lattice latter realization is again a lossless system. And it has to be, because it produces the y tilde output from the u inputs and the y tilde output corresponds to the H tilde the transfer function. And we have actually designed the H tilde the transfer function to be lossless. Remember we designed H tilde as A(z) divided by A tilde (z) with the reversed of the filter coefficients etcetera. And then we decided H tilde was indeed a single input single out which lossless system, so an all pass system. So, a few slides back we designed H tilde to be all pass. This is the frequency domain view on losslessness. And here we see in the realization that this H tilde apart is indeed realized by means of orthogonal rotations only together with delay elements. And so you see the losslessness from the realization actually.

As we have done it for other IIR filter realizations, you can check what happens in the particular case when all of the a coefficients are equal to zero, because then the H(z) transfer function reduces to an FIR transfer function. Now, if all the a coefficients are zero, you can go back to the one page of mathematics and find out that the first reflection coefficient and then the first sine of a rotation angle is going to be zero. And then if you continue with the same mathematical derivation, you will find out that actually all reflection coefficients and so all the rotation angles are indeed equal to zero. And this is the case I focused on two slides back if all the rotation cells have a rotation angle equal to zero and all the rotations basically become void and the lattice part reduces to a pure delay line. So go back to slides and verify that in this case indeed the lattice section corresponds to a pure delay line and then the delayed signals are tapped and multiplied by b coefficients etc etc. This is exactly the way a direct form realization for a FIR system works. So remarkably, we've derived a lattice realization here referred to as the lattice ladder realization and it generalizes the direct form realization for a FIR system. And the way this generalization is done is by replacing the delay line in the case of the direct form FIR system by an

all best system which has the rotation cells etc, which is then the lattice parts. So this is remarkable it is that what we have here doesn't generalize any FIR lattice realization but it generalizes the direct form FIR realization the all pass parts which we see in the lattice realization is also in itself referred to as a 'Gray-Markel' structure. So if you come across the name, this is the realization of an all pass, single input, single output lossless system. So in the lattice ladder realization, the parts that produce the y tilde output from the u[k] input signal and it's lossless and so it's the realization of an all pass filter.

2.29 Slide 46

The last realization that we're going to be looking at for IIR systems is again a lossless lattice realization starting again from the same rational transfer function,H(z), B(z) divided by A(z), and then we're going to, as we always have it, define a second transfer function which is referred to here as H tilde tilde of z. And then of course we have to define the H tilde tilde of z, the way H tilde tilde of z is defined in this case is as follows starting from the denominator. Polynomial A(z), the denominator polynomial is the same denominator polynomial as you had it for H(z), right? So with the same a coefficients then the numerator polynomial to b tilde tilde of z polynomial with coefficients b tilde tilde. The way we define these in this case is by relying again on the magic equation. So we're going to define the second transfer function h tilde tilde such that h together with h tilde tilde satisfy the magic equation. And we know what the meaning is of the magic equation. It means that h and h tilde tilde are power complementary. And so we are going to form a lossless one input two output system. Based on this equation, if you substitute the expressions with B and A and b tilde tilde for H tilde tilde, you find an expression basically in equation from which you can, similar to the way we had it in the FIR case, from which you can compute the coefficients of the b tilde tilde polynomial. So again, similar to what we had in the FIR lossless lattice realization. We define a second transfer function H tilde tilde satisfying magic equation the meaning of the magic equation is that the two transfer functions form a lossless one input, two output system, and there is indeed a simple procedure to compute the coefficients that to be tilde tilde coefficients, in this case by relying indeed on the magic equation.

2.30 Slide 47

The starting point for the derivation of the lossless lattice realization is again the direct form realization of the two transfer functions. What we have here is a system that takes input signal u[k] produces the corresponding to output signals y tilde tilde of k and y[k]. It's a feedback system where the two filters that I realized here together share the same feedback parts where the a1, a2, a3, a4 coefficients and then in the forward parts you see bottom line of multipliers to see the original coefficients to b zero, b one, b two, b three, b four and then the other line of multipliers. In the feed forward path, you see the b tilde tilde parameters that have been defined in the previous slide. So now we're going to do an order reduction procedure applied to this. And again, it will take one page full of mathematics.

So the end result of the repeated order reduction step is indicated in this slide and it's referred to as the lossless lattice realization. It's a realization that so again takes the same input signal u[k] and produces two output signals y tilde tilde, y[k]. Consists of lattice sections and each lattice section has orthogonal rotations. And in between the latter sections, you see signals flowing from left to right in the top line or top branch. And then? At the far right of the slide you see, the signal appearing there bounces back into the lower branches. In the lower branches you have signal flowing from right to left, and some of the intermediate signals are delayed and other signals are not delayed altogether. You see the lattice sections, the signals flowing from left to right. In the top branch and signals flowing from ride to left in the bottom branches with the delay elements as many delay elements as you have lattice sections. So that is the order or that is defined by the order of the original transfer functions. Each lattice section has orthogonal rotations, the orthogonal rotations in the specified lattice section here, theta0 and sine of it. And you should notice the details of the rotations and how they apply to input signals and produce output signals. So the first rotation which is applied here is the theta0 rotation. And it basically takes

the input from the top left and an input signal, intermediate signal that flows in at the bottom right, rotates these together to produce two output signals, one output signal at the top right, which is propagated into the next lattice section. And one output signal at the bottom left, which is an intermediate signal inside this lattice section. This intermediate section is the signal is then rotated together in the second orthogonal rotation with the sines of zero rotation angle. The second input signal comes in at the bottom right, which is an intermediate signal produced by the next lattice section, so these input signals are rotated together. Producing two output signals to the right, and in this case it's the output signals by y tilde tilde and y, for other sections, it's intermediate signals that are propagated to the next lattice section so this is the lossless lattice realization altogether.

2.31 Slide 48

Few things that can be noticed here about this lossless lattice realization is that each section as orthogonal transformations only. So each section with the two rotations basically takes three input signals and rotates these together through two orthogonal rotations to produce three output signals and as you apply two orthogonal rotations, again you can say that the sum power of the input signals is preserved. So the sum power of the three output signals is equal to the sum power of the three input signals. Complete lossless lattice realization has only such orthogonal transformations, plus delay elements which are themselves also lossless. So altogether this again has to be a lossless system. So you see the losslessness in the realization again because you see only orthogonal rotations and delay elements and this is the way the transfer functions indeed have been designed. You defined the H tilde tilde a few slides back from the H by having H tilde tilde together satisfy the magic equation, which indeed means that the two transfer functions together for a lossless system. It's interesting again to do the exercise when all of the a coefficients are zero, so then your original H(z) is reduced to FIR transfer function. In that case, you can easily check all of the theta rotations, as you had in the previous slide, are equal to zero. That means in the propagation, the signal propagation from left to right, the signal is basically untouched. And so the input signal u[k] propagates to the far right and then bounces back and it's still the same input signal at that point. And then in the propagation from right to left, you have the the other rotations with a sine angles. And this is what you end up with then is actually the fir lossless lattice realization. So this IIR lossless lattice realization indeed generalizes the fir lossless lattice realization. The similar special cases were all the sine rotations are equal to zero. And this is where you can check that the complete lossless lattice realization. Reduces to the Gray-Markel structure as we have to find a few slides back. This is where the y[k] output is basically zero and the y tilde tilde the output is an all pass filtered version of the input signal u[k], you can easily verify that.

Final slide here is final remarks that the lossless lattice realization, as we have derived in the previous slide, can also be generalized to a system with one input and N output. So here again, all of the corresponding transfer functions together have to satisfy a magic equation. Go back to the fir case to see an example of that with one input and three outputs, three transfer functions together satisfying a magic equation. If you do the lossless lattice realization, in the IIR case, you're going to see lattice sections with a structure which is indicated in this slide. So you have in this system with one input, three outputs. Each lattice section is going to have three rotation angles. And so implement three rotations, the first rotation in this case, in this specified section is rotation with theta0 that applies to the initial input signal u[k] and produces an output signal that propagates in the top branch to the next lattice section. Then it also produced the same rotation also produced intermediate signal, which is then rotated with the signals flowing in the lower branches from right to left, and that corresponds to the rotation with theta2 first and then theta1 producing output signals or intermediate signals which are propagated into the next lattice section to the left. So the lossless lattice realization indeed can easily be generalized into a more general structure with one input and N output. In this case, in this ends the lecture on filtered realizations.

3 Chapter 6

Chapter 6

3.1 Slide 4

In the previous chapter we looked into filter realization and we saw many different ways of realizing one and the same filter transfer function. And there you could ask yourself the question, why do we bother? Why would we be interested in many different realizations for one and the same filter with one realization not be sufficient? And in this chapter, this is basically where you will find the answer to that question.

3.2 Slide 5

Basic filter implementation or finite word length problem is this. So far we have assumed that all the filter coefficients and the signals are represented in infinite precision and that also the arithmetic operations, the additions and multiplications are performed with infinite precision. In practice, of course, numbers have to be represented in the computer memory and are represented only to finite precision and so filter coefficients, signals and arithmetic operations are subject to quantization error. The analysis is relevant when you consider fixed point implementations, for instance with very short word lengths, which for some applications could still be relevant for other applications where you use long word links with 24 or whatever bits, or where you apply Floating Points arithmetic. The analysis in the chapter here is obviously less relevant. In the rest of the chapter we will investigate the impact of quantization of filter coefficients and then quantization of signals and arithmetic operations.

3.3 Slide 6

Here's first a very simple example, the filter which has been designed here. So in Matlab, it is an elliptic low pass filter and what you see plotted here is the frequency response magnitude or frequency response and then a pole zero plot. As you see, the zeros of the filter basically lie on the unit circle and zeros pull down the frequency response so set the stop band response of the filter. Whereas the pulse of the filter here we're very close again to the unit circle and pull up the frequency response so the poles across the right-hand side plot corresponds basically to the peaks and the magnitude response as you see it here. So this is our filter and let us now analyze different implementations of that filter.

3.4 Slide 7

For this filter, we're going to compare different implementations based on different realizations and the realizations that we consider are on the one hand to direct form realization and on the other hand the lattice ladder realization. In this slide, we compare an infinite precision implementation of the direct form realization against an infinite precision realization of the infinite precision implementation, I should say of the lattice letter realization. In infinite precision, obviously, all realizations should produce the same output signals when you apply the same input signals. If we applied an input signal, what is plotted here of the output signal from the two implementations and the difference between the two, as you see, it plotted at the bottom of the slide is obviously zero in the infinite precision implementations.

3.5 Slide 8

In this slide, you see the output signals from two different implementations of the direct form realization on one hand, again the infinite precision implementation on the other hand, see here 8-bit precision implementation where an 8-bit word length is used for all the signals and coefficients in the filters. Here clearly the difference between the output signals is nonzero. The difference is almost as large as the output signal itself, so that makes the implementation quite useless.

3.6 Slide 9

In this slide, you see the output from finite precision implementation of the lattice ladder realization in the middle plot. And this is then compared against the infinite precision output for either to direct form realization or the lattice ladder realization, whichever one you would like to use. Here you see that the difference between the two output signals is essentially 0, or at least very small. Yeah, unlike what you saw in the previous slide. So conclusion from here is that if you are into finite precision implementation, the lattice ladder realization for this filter may work very well, whereas direct form realization for the same filter may not work out that well. So the conclusion is you better select a good realization and this is basically what this chapter is about.

3.7 Slide 11

One more statement is given in this slide and so if you go back to chapter four and you design a filter transfer function, then basically this is in MATLAB filter design provides you with filter coefficients for a transfer function with almost infinite precision, let's say 15 decimal digits precision. That is the filter you have designed which is supposed to meet your specifications and which you then use for the realization and the implementation and in some of the realizations, you will see the filter coefficients appearing. The A and the B coefficients for instance in a rational transfer function appear in a direct form or transpose direct form realization. In other realizations you will use sort of derived filter coefficients derived from these A&B coefficients, like reflection coefficients. Such in lattice realizations, for instance, it still computes through almost infinite precision. Then you go to implementation and then you have to basically quantize your filter coefficients appearing in your realization to finite precision to the word length used for the implementation. And that in fact leads to a filter that you're implementing, which could be different from the design filter designed in chapter 4 in MATLAB. And the implemented filter may then actually fail to meet the specifications. And this is actually what we're going to be studying in the next few slides. To start with a simple example, here's an example from a reference. As it is cited here, it's an elliptic bandpass filter of which you see as a pole zero plot to the to the left of the graph. The circles are the zeros, the crosses are the poles. You see zeros lying on the unit circle that define the stop bands of your bandpass filter and the poles or lying close to the unit circle and basically defined the band-pass filter. If you then apply or if you then implement this filter with finite precision with 16-bit precision for the filter coefficients in a direct form realization for instance. It turns out that for the implemented filter some of the poles may have even moved outside the unit circle, which is bad news of course, because that turns your filter unstable and then completely useless.

3.8 Slide 12

Two slides ago we looked into a second order polynomial, the nominator polynomial of a second order rational transfer function with the gamma and delta coefficient. And then we looked into the effect of quantization of the gamma and the delta and what it led to in terms of the position of the roots of the polynomial and hence the poles of the system as a whole. And the conclusion was that it didn't look very good. And in general, the fact is that indeed for especially high

order polynomials, the roots can be very sensitive to small changes in the coefficient values. A very extreme example is the so-called Wilkinson polynomial. The polynomial specified here is a 20th order polynomial of which the roots are equal to 1 2 3 4 5 up to 20 right as it is plotted in the left-hand side plot. And then it turns out if you change one of its coefficients only very slightly, then immediately the poles or the roots I should say, of the polynomial move into completely different places. So very indeed you see that especially in the higher order polynomials, the roots are very sensitive to changes in the coefficient values of the polynomial.

3.9 Slide 13

In fact, the previous slide can actually be analyzed and analysis that we're not going to do here, but the result of the analysis, first order analysis is basically stated here and it goes something like this. You start from an initial polynomial and initial polynomial has coefficients A1 A2 up to AL for an Lth order polynomial and for this polynomial you can compute the roots and the roots are given here as P1P2 up to PL. Now from this polynomial you derive a second polynomial which is referred to as a quantized polynomial, where you basically quantize, or let's say just change the filter coefficients from a1 into a1 hat, from a2 into a2 hat, etc. So you have a different quantized polynomial for which again you can compute the roots and these roots are referred to as the quantized roots and are given here as P1 hat P2 hat up to PL hat. The fundamental question is how far is P1 hat from P1, how far is P2 hat from P2 at etc. And this is basically given by the final formula. It says for the Lth root the distance between PL hat and PL is on the right-hand side of the equation and in the right-hand side of the equation you see that you have a summation over I which basically sums the individual effects from the changes of the individual polynomial coefficients. So you look into the change of the ith coefficient. So this is ai which is changed into ai hat. And so this change in the ith coefficient has an effect on the change of the Lth routes and this effects actually has a factor and this is the important part. Or I should say the important part is that in this factor, if you look into the in the denominator of the factor which you see appearing there is a product of a distance between the Lth route that you're analyzing and all other routes that you have. So you have a product PL minus PJ for all possible J's different from L. So this is all possible distances between the LTH route and all of the other routes. Now the observation is in a high order system. You're going to have many poles or I should say in high order polynomial you're going to have many roots and in a stable polynomial if it is the denominator of rational transfer function, all these roots are to lie inside the unit circle. And of course there's not much room in just a unit circle. So if you have many poles and you have to put all of them inside the unit circle, the distance between some of these poles is going to be obviously very smart. And so if you have a very small distance between a pair of roots, then you have a very small number and this product that you see in the denominator of that very factor. And something small in the denominator leads to a very large factor in a very large amplification effect of a change in an individual coefficient of the polynomial leading to a large change into the position of the poles when you go from the poles to the quantized poles. So that again confirms that in the high order, polynomials changes in the coefficients of the polynomial may have a dramatic impact in terms of the position of the roots of the polynomial, and hence if you go to realization and implementation, there's a strong preference for low order systems. This is also basically the motivation for looking into a parallel cascade realizations rather than realizing high order rational transfer function just like that with direct form or transpose direct form realization for instance.

3.10 Slide 14

To develop a bit of a feel for the problem, here's a second example with a simple second order system for which we analyze the effective coefficient quantization on the pole locations. So in this simple transfer function with alpha and beta in the numerator polynomial and gamma and delta in the denominator polynomial, it's the quantization of the gamma and the delta that will have an impact on the actual pole location. And so this second order system could be realized

and implemented. As a second order system and if you use a direct form or transpose direct form realization, and in the realization, you're going to see the gamma and the delta appearing. Or the second order system could also be part of a cascade realization. And then again if you use direct form or transpose direct form realization for the parts and the cascade again you will see the gamma and delta appearing in the realization and be subject to quantization. So the question is as a function of the gamma and delta and the quantization of the gamma and the delta, where are the poles of this second order system? Now for this system you can easily do a stability analysis which allows you to decide which players for the gamma and delta actually leads to a stable filter with stable roots for the denominator polynomial in the transfer function and the conclusion which is immediately drawn if you actually apply the sure call. Stability test, as we have seen it in the previous chapter, is that for all combinations of the delta and the gamma that lie within the so-called triangle of stability, which is plotted here. If you're going to have a stable second order system, the poles of your system, the roots of the denominator polynomial, are going to lie inside the unit circle. So you have to pick a gamma and a delta. So the combination of those lies inside this triangle of stability.

3.11 Slide 15

Now look into finite word length effects or quantization effects. When we quantize the filter coefficients, the gamma and the delta and we'll look at where, we consider a very extreme case where we use only 5 bits for the representation of the gamma and delta and five bits basically corresponds to 32 possibilities. So we consider 32 possible values or possibilities for the gamma. And from the previous slide we know that the gamma has to lie between -2 and 2, so 32 possibilities from -2 to 2 and similarly for the delta 32 possibilities from -1 to 1. And then we consider those combinations of the gamma and the delta that lie inside this triangle of stability. Hensely, two poles or correspond two poles that lie inside the unit circle, and we plot those poles and that leads to the plot as you see it in the right-hand side figure. So what you see here is the complex plane and the unit circle and every cross corresponds to a possible pole position which we would refer to this as quantized poles that corresponds to quantized versions of the gamma and the delta. Once you observe is that you do not have a uniform distribution of the quantized poles within the unit circle. You have regions where you have a high density of quantized poles and you have regions where you have a very low density of quantized poles. And this is especially the case around Z is equal to 1 on the real axis and Z is equal to -1 also on the real axis, where you have a very low density indeed of quantized poles. Now if you design a low-pass filter, you would actually want a particularly narrow-band low-pass filter. You may want to put your poles in the neighborhood of z = 1. But the plot here tells you that this is not going to be possible. So you design your low-pass filter and the position of the poles will lie close to Z=1. But then if you quantize the gamma and the delta coefficient, basically the pulse will jump to the, let's say nearest cross, which could be in a sense quite far away. And so when you quantize the gamma and the delta. The message is that it's going to bring quite a change in the transfer function and hence in the behavior or specification of your filter. And you've started from specifications of your filter and your initial infinite precision design satisfied those specifications. But it could be that after the coefficient quantization you indeed no longer satisfy these filter specifications. And same example for a high pass filter where you may want to put your poles in the neighborhood of Z = -1.

3.12 Slide 16

A possible remedy here is to use a different realization, and the realization which is used here is referred to as a coupled realization, and it's the realization drawn on the slide here. You know what you see in the realization or what you recognize is again a second order system. It has two delay elements, so this is second order. It has feedback. So it's a feedback system with an infinitely long impulse response. So that corresponds to a rational transfer function. You see filter parameters appearing here. These are the , , . And wherever you see such a Greek letter, the meaning is actually that you multiply a

signal by this filter parameter, and so these parameters are these filter coefficients. I should say the , , are related to or can be computed from the alpha, beta, gamma, and delta as you had it in the rational transfer function in the previous slide. So you could go out and design your filter in Matlab if it gives you a transfer function with alpha, beta, gamma and delta, and those four filter coefficients are then converted into new filter coefficients , , to give you with this realization and input output behavior which is still described by the same rational transfer function with alpha and the beta and gamma and delta. Now the observation here is that for this realization the poles exactly lie at new plus or minus j*. So if you go out and quantize the filter coefficients now, for instance and , you're actually immediately quantizing the position of the poles of your system. And that leads to a plot as you have it in this slide compared to the plot in the previous slide, where within the unit circle you're actually going to see a uniform distribution of quantized poles, which is much more favorable. So this time we go out and design your filter in MATLAB. Rational transfer function with the alpha, beta, gamma and delta. It converts these four filter coefficients into new filter coefficients , , . You do the realization, quantize the filter coefficients and it leads to sort of a change in the actual pole position. You jump to the nearest cross, but everywhere you're going to have in your cross, and it's not going to have a dramatic impact on the filter that you're actually realizing. If the originally designed filter satisfied your specifications, with high probability that the implemented filter will also satisfy your specifications.

3.13 Slide 18

Message from the previous slides is that when your design a filter and you use a high order rational transfer function for an infinite impulse response filter and you would implement the filter based on a direct form or transposed direct form realization where you immediately see the coefficients of the denominator and polynomial of your transfer function sitting in your realization and then these coefficients will be subject to quantization. That is really a normal, because the quantization of the filter coefficients is going to have dramatic impact on the position of the poles of your system and so the input output behavior of the implemented filter could be completely different from the input output behavior of the originally designed filter and perhaps no longer satisfies the initial specifications from which you start your design. So the better alternative is to use other realizations, for instance cascade and parallel realizations where you split your high order system into lower order systems, it's a second order systems and then for each second order system you could think of using a direct form or transpose direct form or perhaps a couple to realization as we have seen it earlier up in the previous chapter. We've also discussed alternative realizations like lattice realizations and then the question could be whether these realizations perform well under coefficient quantization. Here we're not going to do any further analysis just to put a statement without sort of a formal proof. But in the case of lossless lattice realizations, where all of the coefficients are sines and cosines, and hence you can already see that these are well behaved dynamic range of all the coefficients is basically between -1 and 1, so it's easily controlled. You can observe and also analyze that in such lossless lattice realizations, coefficient quantization does not have or is never going to have such a dramatic effect, and there's a small example included here of an original transfer function. And then they realize the transfer function and direct form and quantize the coefficients and analyze what or how the transfer function changes on their coefficient quantization do the same thing based on a lossless lattice realization, and then observe that in the latter case the transfer function hardly changes, whereas in the case of the direct form realization on the coefficient quantization we see a dramatic change of the transfer function.

3.14 Slide 20

Problems for this part are given in this slide. So now we're going to consider signals, the overall filter output signal as well as internal signals inside the filter realization of implementation. And for these signals we're going to consider finite

word length representations again. Obviously, a finite word length implementation implies a maximum representable value or number, and so whenever a signal exceeds this maximum representable number, you have an overflow condition. Overflow can be very harmful and can lead to polarity reversal for instance, so it is to be avoided. And then an obvious way of avoiding overflow is just to use more bits in the finite word length representation. But then in your implementation, typically you have sort of a target in terms of the number of bits you would like to use in the implementation. You wouldn't want to go on increasing the number of bits in your implementation, so you want to limit this. And so the question is how can you still avoid overflow without increasing the number of bits? And the way to do this is to introduce proper signal scaling, which is basically equivalent to first increasing the number of bits, then removing a number of least significant bits, for instance the 10 least significant bits to have the same word length. So when you do so. When you scale signals like this, each signal will have an implicit scale factor, which is denoted here as 2^C, where C is basically the number of bits, low significant bits that you have removed into previous bullet so that the true signal value is actually 2^C times the number which is represented by the word of bits reserved for that signal and then the question is basically how do you set these scale factors what would be a good strategy for setting these scale factors.

3.15 Slide 30

Problem statement for this part is indicated in this slide. We're considering addition and multiplication, so addition first, when you add 2 numbers to samples of two signals, you're adding a first number with B1 bits, let's say in a scale factor 2^C1 then adding that to a second number which is for instance would be B2 bit number with a scale factor 2^C2 and that results in a number with a number of bits that is indicated by the somewhat complicated looking formula here and with the corresponding scale factor. And similarly when you multiply 2 numbers together you would start from the same B1 bit number with scale factor 2^C1 and B2-bit second number with scale factor2^C2. And the result in this case is simpler. This is a B1+B2-1 bit number with scale factor 2^(C1+C2). So in both cases you end up with a number with a number of bits and a scale factor. But that has to be matched to the number of bits and the scale factor which has been reserved for the corresponding signal or result of the operation. And typically that requires removing a number of least significant bits because typically the arithmetic operation gives you more bits than what you would actually like to see. So typically this matching operation results in removing a number of least significant bits, which can be done by means of a quantization mechanism such as rounding or truncation, but in any case introduces quantization noise, and it's this quantization noise which is introduced after every addition or every multiplication. The effect of that quantization noise is eventually on the output of the implemented filter. This is the thing we would like to analyze here. In the next few slides you will see a first statistical analysis of the quantization noise which is rather straight forward. But then the observation is that the process is really not a statistical or stochastic process, but it's purely deterministic. But they're nonlinear and difficult to analyze. But you can somehow estimate what the result of this can be by running simulations, and this is where you're going to observe things like limit cycle oscillations and such what will be discussed in later slides.

3.16 Slide 31

When you have to remove a number of least significant bits, you have to apply a quantization mechanism. In this slide a few popular quantization mechanisms are specified or illustrated. In the graphical representations you see an input and an output graph. The input is the input signal to the quantization mechanism. So the first result from an arithmetic operation, let's say, and then the outputs the outputs of the quantization mechanism. The bold line is basically the quantization function, and it's always a staircase type of function where the steps correspond to the numbers that you can represent with the fewer bits after you have removed to the least significant bits. The thin line is the line where the output is equal to the input, which is just added for reference. Then the distance between the thick line and the and the

thin line is basically the quantization error as you introduce it by the quantization. The three mechanisms which are indicated here are rounding, truncation, and magnitude to truncation. In rounding, you round towards the number that you can represent with the fewer bits which is closest to the inputs of the quantization in truncation. You truncate to such a number, which is always smaller than the input to the quantization, whereas in magnitude truncation you always truncate towards zero, so that the output of the quantization is always smaller than the inputs to the quantization, smaller in absolute value. Now the distance between the bold line and the thin line is the quantization error. There's a statistical way of specifying this quantization error, at least if you assume that the input to the quantization operation is uniformly distributed. Basically, you would say that the input signal sits on the horizontal axis everywhere with the same probability. Then you could specify the quantization error. And then for instance for rounding you would decide that the quantization error on average is zero, so the mean is equal to 0 and the variance is given by the expression here on given on the slide, which you can easily check actually for truncation. The mean is -1 half and then the least significant bits that you're removing if in the case you're having just one least significant bit with the corresponding variance as it is also indicated here, and then finally for magnitude truncation mean is again equal to zero and then with a certain variance.

3.17 Slide 32

To analyze the effect of the quantization operations after the arithmetic operations on the overall outputs of the implemented filter, we're going to do a statistical analysis, and this statistical analysis will be based on three simplifying assumptions, which are listed here. The first assumption is that each quantization error is random, uncorrelated independence of the number that is being quantized. So if you go back to the previous slide, the quantization error is the distance between the bold line and the thin line that is assumed to be uncorrelated independence of the inputs to the quantization operation. And then if you assume that the inputs to the quantization operation is uniformly distributed over the horizontal axis, let's say in the previous slide, that allows you to specify a mean and a variance for the quantization error as we have done it in the previous slide. The second simplifying assumption is that successive quantization errors at the outputs of a multiplier/adder. So after a multiplication or addition operation, errors are again uncorrelated, independent. So one multiplier, one adder followed by a quantization operation produces a sequence of quantization errors and this is like a white noise sequence. Third simplifying assumption is that quantization errors produced at the outputs of different multipliers/adders are again uncorrelated, independent. This is the independent sources assumption. These assumptions are simplifying assumptions. You can question the validity of the assumptions. For instance, think of the second assumption and imagine you have a multiplication operation where the input is a DC signal. So all samples are always the same and you multiply these samples by always the same filter coefficient. Then the result of the multiplication is always the same and hence the quantization error will also always be the same. So that's basically DC quantization error signal so that’s not a white noise sequence. So you can question the validity of these resumptions and conclude that basically, in practice these are not valid assumptions. Nevertheless, they allow their simplifying assumptions, as I called them, and they allow for simple statistical analysis that indeed gives an idea of what the impact of all the quantization errors is going to be and the output of the implemented filter. So by exploiting or using these assumptions you can follow the strategy, something like this. You consider your filter implementation and then wherever you have an arithmetic operation like addition or multiplication you view this as like an infinite precision operation which is then followed by a noise insertion which corresponds to the quantization noise. So you add a noise source after each addition or after each multiplication in the block scheme. You have a simple example here with the feedback filter, where after the multiplication with the filter coefficient -.99 you have a noise source added which is the e1 and after the addition at the top of the scheme you add in a second noise which is the e2 as you see it here. And now of course we're dealing with a linear filter structure. So if you look at the output signal, the output signal is going to have a contribution from the input signal u[k]. So the output y[k] will have a desired contribution from the input signal u[k], and then we'll have additional contributions which

can be easily computed separately from the e1 as well as from the e2. And it's these two additional contributions that have to be analyzed and compare to the desired contribution from the input signal to the output signal.

3.18 Slide 33

The effect of an individual noise source where you inject a sequence of quantization errors on the outputs of the filter will be analyzed in terms of a mean and a variance. And to do this you follow a very simple procedure which is indicated in this slide. So you have a noise insertion, a noise source at some point in your filter and it inserts a sequence e[k] which is assumed to be white. So a white noise sequence with a mean and a variance that $_e$ and the $_e^2$. And if you go back to the slides you see how from a specified quantization mechanism you indeed can specify a mean and a variance for the quantization error. And then from the noise insertion point, the point where you have the noise source to the outputs of the filter, you can define transfer function which is denoted here as G(z) in the Z domain or G[k] as an impulse response sequence. This transfer function is for the noise source and then if you're given a mean and a variance of a signal at the input of a transfer function, then the mean and the variance at the output of the transfer function, so at the output of the filter is easily computed. The mean of the quantization noise in the output signal will be the mean of the inputs of the noise transfer function, which is the $_e$ which gets multiplied by a factor and the factor is the DC gain of the noise transfer function. So, you evaluate G(z) for DC, so e^{j0} which basically is equal to 1. Similarly, the noise variance at the output is the noise variance at the input of the noise transfer function multiplied by a factor which is now equal to the noise gain. And the noise gain is basically the squared L2-norm of the impulse response sequence, and this is also a well-known formula. So, this allows you to analyze what the output mean and variances of an input mean and variance for a certain insertion point for a certain noise source that you have added in your filter structure. And the strategy is to insert a noise source after each arithmetic operation, multiplication or addition. So, you will have to do this analysis, repeat this analysis or repeat this procedure for each and every arithmetic operation or noise source that you have inserted after each arithmetic operation. And then the overall quantization noise, mean and variance at the output of the implemented filter will be the sum of the means and the sum of the variances.

3.19 Slide 34

The analysis as you have seen in the previous slide is straightforward and simple. At the same time, it's a bit tedious because you have to repeat this analysis for each and every arithmetic operation in your implemented filter. Luckily, for some filter realizations the analysis can be significantly simplified. For instance in a transposed direct form realization, for example, you can check that if you insert a noise source after all the multiplications with the b coefficients and as well as with the a coefficients and also after all the additions, the horizontal line in the in the middle, you would see that for all these noise insertion points you would end up with exactly the same noise transfer function up to at most delay which corresponds to the delay line in the middle of the scheme. So all noise transfer functions are basically the same up to a delay and that means that all DC gains and noise gains as you headed in the previous slide are exactly the same number. And the result of this is that you can basically lump all these quantization noises into one big one quantization noise which has one insertion point as it is indicated in the slide here. And so you have to do the analysis only for one noise insertion and it will turn out that this is then a very simple analysis.

3.20 Slide 35

Second example is for a direct form realization, where you would see that if you add noise insertions after the multiplications with the A coefficients and after the additions in the top row of the realization that all of these have again the same

noise transfer function and hence can be lumped into one quantization noise insertion, which is the e1 at the top and similarly for all of the multiplications with the B coefficients and the additions in the bottom row. Again, you will have the same noise transfer function and hence these noise insertions can be lumped into one noise insertion that you could do two at the bottom of the slide.

3.21 Slide 36

Statistical analysis from the previous slide is very simple, very convenient and allows you to get an idea of sort of the size or the impacts of the of all the quantization and a filter implementation impact in the overall output of the implemented filter. Nevertheless, you can question the validity of this statistical analysis. A few slides back I had basic assumptions on which the analysis is based and there I already gave an example where you see that these assumptions do not hold true in practice. Think of a multiplication operation for instance, which is fed with a DC signal where all the samples are the same number and you multiply it with a fixed filter coefficient. The output from the multiplication is also a DC signal, always the same samples, hence always the same quantization operation with the same quantization error and hence the quantization error signal is also a DC signal. So nothing like a white noise sequence. So all of these assumptions as you had at a few slides back can be questions and in practice they will not hold true. What you do have in practice is a nonlinear system. If you go back to the slide with the specification of the rounding and truncation and magnitude truncation, you see the bold lines for the staircase functions, which are basically nonlinear functions. So what you do have is a nonlinear function, nonlinear but deterministic. So there's nothing stochastic or statistical or probabilistic about such a system. If you have a filter implementation with quantization in place, you would run the filter today with a certain input signal, produces certain output signal, including quantization effects. You run the same filter tomorrow with the same input signal, produce exactly the same output signal with nothing random in it. The whole filter is a non-linear deterministic system. Obviously nonlinear systems are difficult to analyze theoretically, but what we can do instead of throwing up an extensive theoretical analysis, is run examples to see what the behavior of such nonlinear systems could be like. And to get a feel for this, here's a very simple example. As I said, you can run simulations in MATLAB for instance with finite precision. So this is an example of a very simple first order feedback system specified by a difference equation. The output sample at time k is –0.625 times the previous output sample plus an input sample u[k] and we implement this filter with four bits, a very extreme case of course, here 4 bits arithmetic with rounding after the multiplications and after the additions. Now if you run a simulation where the input signal to the filter is uniformly equal to 0, so there's basically no inputs to the filter and you start the filter with the output sample at time zero is 3/8. Now as the input to the filter is zero and the filter can be checked to be a stable filter. The output should obviously converge to zero, but when you run the simulation, you will see that the output sequence is something like what's indicated here, so 3/8, etc. But then after a while you see something appearing, which is referred to as a limit cycle oscillation, where you see an output sample 1/8, -1/8, 1/8, -1/8 etc. This is an oscillation, and it's referred to as a limit cycle oscillation. So the output of the filter converges in the limit to such a cycle oscillation, and it's referred to as a zero-input limit cycle oscillation because indeed the input signal is 0 and so as the input signal is zero in a stable filter, the output is expected to converge to 0, but it doesn't. So the output lives on without there being an input signal, which is definitely undesirable. And this is zero limit cycle oscillation behavior which is observed in a nonlinear system, a filter implementation with quantization effects.

3.22 Slide 37

The zero-input limit cycle behavior obviously has to do with the quantization that we have in the filter, so you can check if it goes away if we apply a different quantization mechanism. Here's a second example, same filter, but now we do 4-bit truncation instead of 4-bit rounding. And then you observe that indeed the output sequence of samples converges to

zero, as you would expect it to do. But then it turns out that this does not always work. Truncation does not always make the zero-input limit cycle oscillation behavior go away. Third example here, we have a filter where the difference equation that -0.625 has been replaced by +0.625. Still a stable filter. When you do 4-bit rounding, your output gets stuck in 1/8 which is not the idea. If you replace the rounding by truncation, the output gets stuck in -1/8. So everything here is rather weird. But what you do observe is in the sense strange and very undesired behavior of your nonlinear system due to the quantization mechanisms as you have introduced them.

3.23 Slide 38

Limit cycle oscillation behavior is definitely not very good. It means your output signal of an implemented filter sort of lives on when the input signal is already zero. So imagine an audio system when you switch off the music, the loud speakers in a sense keep going. That's definitely nothing you would want to see. Now you can easily see that limit cycle oscillation can happen and filters that have feedback and feedback filters are filters with an infinitely long impulse response like IIR filters. Whereas when you consider FIR filters, finite impulse response filters that do not have feedback, for these filters you will never have limit cycle oscillation behavior. And an FIR filter the impulse responses is finite so the finite end time, so the effect of quantization errors is also finite in time. So if the input is set to zero, so after a number of samples corresponding to the length of the impulse response, the output will also be zero no matter what type of quantization you have. So here again you see that in practice, if IIR filters never ever give you trouble, whereas IIR filters, feedback filters can indeed give you a lot of trouble, then limit cycle oscillation behavior. The mathematical analysis for this is very difficult and so we're not going to do this. This is nonlinear systems analysis. We're not going to go into that.

3.24 Slide 39

Now here’s the good news. The good news is if you use a good realization, then actually you're guaranteed never to see a limit cycle oscillation behavior. So we're discussing IIR filters here, and the good visualizations for IIR filters are the lossless lattice realization and then the lattice latter realization and the lossless lattice realization. Remember that only operations of the filter are basically orthogonal rotations, and similarly in the lattice ladder realization, at least the feedback part of the filter realization also again only has orthogonal realizations. Now on the one hand you have to pick the right realization, on the other hand you have to pick the right quantization mechanism. So the good news is actually, if you pick the right realization, lossless lattice or lattice ladder together with magnitude truncation as the quantization mechanism. Then indeed you are guaranteed to be free of limit cycle oscillation behavior. So both condition have to be satisfied. Lossless lattice filters with rounding is not guaranteed to be free of limit cycle oscillation behavior, but lossless lattice realizations with magnitude truncation are indeed guaranteed to be free of limit cycle oscillations. The intuition behind this is that magnitude truncation is specific quantization mechanism. Go back a few slides to see that the output of the magnitude truncation is always smaller because you truncate towards zero is always smaller than the inputs to the quantization operation, so that the power and the output of the quantization is always smaller than the power of the inputs to the quantization. So in a sense your quantization eats up some of the energy or power that sits in your filter and on the other hand you have orthogonal operations in your filter in your lossless lattice or feedback part of the lattice ladder realization and orthogonal operations do not generate any additional power preserve powers. On the one hand you have power preservation and the actual filter operations, and on the other hand you have the magnitude truncation quantization mechanism that consumes some of the power that isn’t sat at your filters. If you have a zero input signal, you cannot feed any power to your filter, and the filter itself doesn't generate any power, only eats or consumes some of the power, then the output signal has to go to zero. That is somewhat the intuition behind the statement here. If you use the right realization together with magnitude truncation, indeed you're guaranteed to be free of limit cycle oscillation. The

conclusion here is sort of a general conclusion, also for the chapter as a whole. So there are significant issues related to filter implementation. But then if you pick the right realization, that can save you a lot of trouble. For instance, if you use a lossless lattice realization, coefficient quantization effects will not be that bad because all of the coefficients are bounded between -1 and 1, because they're cosines and sines. So with a well-behaved dynamic range you can easily sort of monitor the effect of filter coefficients on the actual transfer function that you have implemented. Similarly, because of the cosines and the sines in your filter, the internal signals and the filter implementation are easily scaled. And then as far as the arithmetic operations go, here you see that when you use the proper quantization mechanism, you will never see this annoying limit cycle oscillation behavior. So pick the right realization and it will save you a lot of trouble. That's sort of the main conclusion from the chapter.

3.25 Slide 40

Conclusion for this chapter. Again, you've seen that there are quite a number of issues, non-trivial issues related to filter implementation. So when you use fixed point arithmetic, and especially so with short word lengths, not so much with fixed point arithmetic with long word lengths. Or when you use floating point arithmetic, which has not been looked at here in terms of the filter implementation process for a given realization. The process you could envisage is sort of an iterative process where first you try and set all the word links and the scale factors for the signals and also the filter coefficients in the filter. And then you analyze everything and if you're happy with the result of the analysis, you're happy. If you're not happy, you would reduce as step number one where you change some of the word lengths in the scale factors for the internal signals or filter coefficients. Overall, sort of looking at this Part 2 filter design and implementation process where you have the different steps, first designing a filter transfer function at chapter 4, then picking a filter realization in chapter 5, and then implementing those realizations. In this chapter, you could follow the different steps then analyze. If you're happy with the results from the analysis, you're happy. If you're unhappy, you would revise the filter implementation, or pick a different realization, or redo the transfer function design process. So again, you could sort of envisage an iterative process over all the chapters of this part of the course.

4 Chapter 7

1

This is Chapter 7 on optimal filters, also known as Wiener filters.

2

This Chapter 7 is the first chapter in part three of the course on optimal and adaptive filtering. In this chapter, we'll give a bit of an introduction to the general setup for optimal filters, and then also adaptive filters and illustrate this by means of a number of applications.

And then we'll also give a brief introduction to Wiener filter or optimal filter theory.

Then in the next chapter, we will move from optimal filters into adaptive filters and see two basic algorithms, namely the LMS and the RLS algorithm. Chapter 9 will continue with adaptive filtering algo.

Things where we're going to look into square roots and fast Rs algorithms and then finally, chapter 10, we'll focus on common filters, which is basically the RLS or recursively squares sort of algorithm for dynamic time varying systems. As you will see.

3

Optimal filter design is very different from classical filter design as we have seen it in the previous part of the course where we typically start from a desired frequency response for a filter. So, frequency domain specification of a low pass or band pass or whatever filter and then we train optimally approximate this frequency response. Optimal filter design is also filter design but takes a completely different approach as it is stated here, signals will be viewed as realizations of stochastic processes and the filter design will be done in statistical sense and will be based on E (?) prioritized statistical information and as we go along in later slides, you will see what is actually meant by this and what type of a priority statistical information is actually needed to do optimal filter design in the sense as you will see. The optimal filter design that we're going to consider here is also referred to as Wiener filtering, after Norbert Weiner, the person on the picture here. The basic scheme for optimal filtering is given in this slide already will be repeated in the next slide and explained there.

4

The basic setup for an optimal filter. It's repeated here from the previous slide, and it's explained as follows. We're given a filter input signal, so the 1st signal and when we say this is a given signal, it means we'll have statistical information available about this signal and in later slides we will see what type of statistical information we actually need. Help this first filtering book signal to do the optimal filter design. And so, the filter will be fed with this first filter input signal. The given signal will produce a filter output signal, and then the idea is that the filter output signal is optimally close to a second signal which is referred to here as the desired output signal or the desire to signal. So, there's a second signal of which again, we'll need statistical information and again, it will turn out what type of statistical information about the desired signal we're going to need. And so that the overall goal is to have the filter such that when it's filled with the filter input, it produces an output optimally close to the desired signal, and this is also where we'll have to define what we actually mean by optimally close. So, this is where we're going to need a distance measure that we can optimize.

5

In later slides, we're going to see applications of optimal filtering, and then it will turn out that many of these applications, the assumed a priori statistical information about the signals will never be available and so this is where you will have to move from optimal filters to adaptive filters adaptive. Filtering will be the topic of the next and then next and next chapter where basically the prototype adaptive filtering setup is similar to the prototype optimal filtering setup with an additional feature in a sense.

So, what you see in the graph here is an adaptive filter setup. Looks like a setup for optimal filtering where the filter is fed with an input signal that produces an output signal which is supposed to be optimally close to a desired signal, and then the adaptive filtering operation basically involves 2 processes. The first process is the filtering process. You design A filter to filter signal. Yeah. So, basically that's the general operation of the filter as such.

On top of the filtering process you will see an adaptation process where you change the characteristic of the filter in order to have sort of an optimal setting at each point in time. So, this is where you could say if the A prior or statistical information is missing, you have to somehow learn this information from the environment by observing signals and estimating statistical quantities from these observed signals. But it will turn out that it leads to a rather simple scheme where. Basically, the error signal as you see it in the graph here. Which is the difference between the filter outputs and the desired output signal. The difference of those is the error signal that is basically used to steer the adaptation of the filter response or the filter coefficients. You could say so it leads to a bit of or sort of a feedback mechanism. For the error signal is fed back for the filter adaptation, and indeed if the error signal is small then you're happy and there's no adaptation needed. Whereas if the error signal is large, you're not so happy and there's. To where you will have to adapt to the filter coefficients.

6

An adaptive filter with the two processes of filtering and adaptation. As you will see it in the applications in the later slides is actually designed to filter and adapt itself forever. It's initialized. And then runs the filtering and the adaptation and the number of updates or adaptations could be thousands could be millions, could be billions, and nevertheless that the filter is assumed to track time variations, stay numerically stable, and so the idea is that even when you continue doing these updating operations in your filter, you shouldn't be building up numerical errors for instance. So that is meant by your numerical stability, so your filter should be accurate and robust, et cetera at the same time. You're worried about computational complexity and hardware implementation. You should have a filter which is cheap to implement, both in software and in hardware. It's this important. Characteristics of adaptive filters that will be studied in later chapters.

7

In this, in the next few slides, we'll see a number of applications of optimal filtering and then immediately also sense adaptive filtering where the idea is basically to convince you that the basic setup is a valid setup used in many applications, even though initially may look like not a very useful setup where you produce filter output signal that looks like an already given signal. That doesn't look very useful, but in practice it is a useful setup, and you will see it in this slide and in the next few slides. The first sort of family or collection of applications is referred to as system identification or system modeling applications and the basic scheme is provided here. Starting point is a plant and a plant is the term plant is used here to refer to system can be any system as long as it has an input signal and produces an output signal. You see instances of this in the next few slides, so, but assume you're given a plant with produces an output signal when it's fed with an input signal, and to describe or to model the behavior of the plants by using or by means of a mathematical model, we use an optimal. For an adaptive filter which is going to be fed with the same input signal and then of which the filter coefficients will be tuned such that the outputs of the optimal or adaptive filter approximates the output of the plans and if the difference between the optimal or adaptive filter output in the actual plant's outputs, so the error signal if that is small then we can assume that the adaptive filter produces a good model for the mathematical model for the input output behavior of the plants. Adaptive filter we're going to use linear transfer functions. Could be IIR could be FIR mostly FIR as you will see later filters. So that is indeed a mathematical description and difference equation let's say and it's used here to describe to model the input output behavior of the plant.

8

The first example of such channel identification is radio channel modeling, an example that was also used in chapter one actually. So, when you transmit a signal over a radio channel and a simplified representation, the signal is a sequence of ones and 0 bits. That signal is somehow filtered by the multipath radio channel, and so at the receiver which you observe is not sort of simple attenuated version of the transmitted sequence, but it's truly a filtered version of the transmitted signal where you see the differently delayed and differently attenuated versions of the signal transmitted signal superimposed upon each other. And this as we have also mentioned it in chapter one can be modeled by means of a linear transfer function, oftentimes in FIR, Linear transfer function. So here we're modeling the filtering operation by a radio channel and we're doing this by transmitting a training sequence, let's say from the transmitter, from the base station here to the mobile receiver and in the mobile receiver you run your optimal or adaptive filter. So, the optimal adaptive filter is as you see here basically receiver functionality, so the adaptive filter or optional filter is fed with a known input signal which is the training sequence, which is known throughout the network, known to every user in the network. So, the mobile receiver knows with the inputs training sequence to the radio channel is and then sets an optimal or adaptive filter to produce to produce an output signal, which optimally approximates the actually observed received signal in the mobile receiver. And when you manage to do so, then indeed the optimal or adaptive filter provides a suitable mathematical model for the signal propagation in your radio channel, which then as we have also mentioned it in chapter one can be used to compensate for this channel filtering operation through channel inversion or you should say channel equalization.

9

Second example, system identification or modeling example is referred to as acoustic echo cancellation and It's crucial I should say in voice over IP systems like Skype and similar the scenario which is drawn here is as follows. To the right of the figure that is basically you in front of your laptop and your laptop has a microphone that the microphone that records your speech signal which is referred to as the Mirant speech signal or near int the signal. At the same time your laptop has a loudspeaker server. Loud speakers that play the signal which is referred to here as the far end as signal, which is the signal from the other side of the communication link. So again the speech signal is played by the loudspeaker and the thing that can be very annoying. Is that the loudspeaker signal will be picked up again by the microphone because the loudspeaker as we would say is acoustically coupled with the microphone there's an acoustic system in between the loudspeaker and the microphone. It's again a multipath propagation system, where the acoustic signal loudspeaker signal travels through an acoustic enclosure is reflected on walls and objects etcetera and in the microphone. Again you see some sort of multi path filtered version of the of the loudspeaker signal and this is annoying the person at the other side of the conversation. For communication link, I should say basically after a propagation delay will hear their own voice which is first propagated to this side of the communication link played in the loudspeaker and then picked up by the microphone and sent to the other side of the communication link you can so that acoustic coupling between a loudspeaker and a microphone in the near end room leads to an annoying echo effect in the which is perceived or observed in the other room in the far end room, and this is annoying. This sort of disturbs natural conversation. It should be counteracted and the way to counteract this or compensate. Or this echo effect is to use again an optimal or an adaptive filter so as you see it here, the adaptive filter or optimal filter. This is implemented in the laptop in the near end room. So in your laptop, it takes as an input signal the loudspeaker signal and it produces an output signal which optimally approximates the microphone signal for the tuning of the adaptive filter. Assume that the near end signal is not active so that the microphone signal only picks up the Echo loudspeaker signal to keep things simple and so this is where you can tune your optimal filter or tune your adaptive filter and when it manages to produce an output signal which approximates the microphone signal, then you can actually subtract that signal from the microphone signal and what results is an error signal, which or in which the ECHO contribution is completely cancelled. Even when then your end signal becomes active. So that is briefly the operation of an acoustic echo cancellation as needed in a voice over IP system and what you see here is that indeed the optimal or adaptive filter provides a mathematical model. In this case for the signal propagation. And an acoustic channel radio channel in the previous slide here similarly, but now an acoustic channel from the loudspeaker to the microphone.

10

The similar echo cancellation is sometimes used in full duplex modems. This is not an acoustic echo cancellation as you had it in the previous slide, but a line so-called line ECHO cancellation. So in a full duplex modems, think of a full duplex modem connected to network for instance, for fast Internet access. And the modem. Computing a sensor transmit signal which will be transmitted to the network and receives a signal from the network which is then demodulated by the modem. The sort of computation of the transmit signal is done in the discrete time domain, so the result is digital to analog converted and then that signal is transmitted to the network and at the same time from the network you have a receive signal. Which is then analog to digital converted in the lower branch on the in the figure. So analog to digital converter to the discrete time or digital domain where the demodulation takes place. Now the remarkable thing is that you have this two-way traffic to and from the network on the same wire. So, you need some sort of arrangements to organize the transmission on this wire. Together with the reception from this wire. And that is done by an electrical component which is referred to as a hybrid, and the operation of the hybrid is basically to take the signal from the top branch appearing from the digital to analog converter and propagating this channel towards the network and at the same time. Receiving a signal from the network and propagating this signal into the lower branch towards the analog to digital converter. So, the hybrid is sort of a traffic control system that takes traffic from the top branch. And propagates this to the right, and takes traffic from appearing from the right, and propagates this into the lower branch. Now it turns out that such a hybrid has to be impedance matched to the impedance of the line. And such a line will never or such a matching

will never be ideal, and as a result of this non ideal impedance matching the operation of the hybrid will be non ideal which results in a part of the signal energy appearing from the top branch leaking into the lower branch. So that means the transmitted signal prepared in the digital domain is the to a converted and then passes through the hybrid. Most of the energy is sent to where it's the network, but some of the signal energy leaks into the lower branch. It's analog to digital converted again and it's received by the modem. So, the modem sort of receives at a certain level some of its own transmit signal, which can be very disturbing for the actual reception of the useful signal from the network. So here you have a similar echo generation mechanism. Now through the DA and AD together with the hybrid and again you have to compensate for this so-called line echo mechanism by means of an optimal or adaptive filter which is fed with the same transmit signal and mimics the echo path or the operation of the echo path to produce an output signal which models the ECHO contribution and the signal in the lower branch and then this is subtracted and hopefully you get a usefully received signal and only a small residual echo. So here the optimal adaptive filter again is used to model align Echo path from the transmitter to the receiver.

11

Next example here is an acoustic noise cancellation example. Imagine you're recording desired speech signal by means of a microphone which is referred to here as the primary sensor or primary microphone. And on top of the desired speech signal, you would record noise signals. Background noise signals. The idea is then to use additional reference sensors or reference microphones which are placed ideally close to the noise source. And then optimally filter these reference microphone signals and subtract these filtered reference microphone signals from the primary microphone signal to get rid of the noise contribution. So in this case you can see that the optimal or adaptive filter in a sense? Models the acoustic path, which is again a multi path propagation system from basically the position of noise sources or the position of the reference. Microphones to the primary microphones. So there's modeling of a number of acoustic transfer functions involved here, but essentially it's again an acoustic path or system modeling exercise.

12

The next modeling exercise is actually a signal modeling exercise. You start from a signal, uk, and basically you're going to use an optimal adaptive filter, and in this case this is going to be an FIR filter. So it basically makes a linear combination of the last so many samples of the signal uk. And you're going to use the optimal or adaptive filter to produce an estimate for the next sample of UK and UK plus one. So the plant in this case you could identify as a one step prediction operation which is denoted here in the scheme by means of Z. So it produces the next sample U at time K + 1 when it's fed with the input signal U at time K and so the operation of the adaptive. Is to set up a one step ahead prediction or predictor to predict the next sample from the last so many previous samples of the input signal, and this is really crucial. This is referred to as linear prediction signal modeling, linear prediction, which and it's crucial it's used in most speech coding, operation speech codecs.

13

Whereas the previous examples were system identification modeling examples here the last few examples referred to as inverse modeling or inverse system identification examples. The idea is basically to consider a plant. A plant again can be any system as long as it produces an output signal from an input signal. So consider a plant fed with a plant input signal, producing a plant output signal. The ideally the operation of the optimal or adaptive filter would then be to reconstruct the input signal from the plant output signal, so the planned output signal is fed into the optimal or adaptive filter, and the idea that would be that the optimal or adaptive filter produces a signal that approximates the plant input signal if the plant input output behavior would somehow be invertible, perhaps up to a delay, which obviously is not always the case, not any linear transfer function has a stable inverse, but if the plant input output behavior would somehow be invertible then can need to envisage sort of an inverse plants taking the place of the plant as you had it in the previous slides and the adaptive filter is fed with the same input signal to the inverse blend and produces or is made to produce an output

signal that approximates the output from the inverse plant, which is then indeed the plant input signal up to at most delay, so this is generally referred to as inverse modeling in the next slide we'll see an example of this.

14

In the previous slide, we've seen how an optimal or adaptive filter can be used in the context of a radio communication to model the operation of the filtering operation of a radio channel where we transmit a training sequence and then use the training sequence to identify.

We put up a model for the radio channel. Once we have this model, we can use the model to design a composition scheme which is generally referred to as the channel equalization scheme, so that is part basically of a twostep approach where you first put up a channel model and then in a second step use the channel model to design a channel composition, a channel equalization. What we have in this slide is sort of a one-step alternative to this. We're here in one step you try to immediately reconstruct the radio channel input signal from the radio channel output signal and that and it is an inverse modeling operation, similar to what you've seen in the previous slide. So here again you consider the same radio channel, the training sequence being transmitted from a base station received in a mobile receiver, and then the operation of the mobile receiver is such that it uses an optimal or adaptive filter to immediately produce a signal that optimally approximates the transmitted signal. So, which in this case in the training phase is indeed the training sequence, so the desired outputs of the optimal or adaptive filter would be the known training sequence known to the mobile receiver and so the mobile receiver chains are optimally designed filter to produce a signal that looks like the training sequence.

15

In the previous slide, the optimal or adaptive filter was used as a channel compensation mechanism or a channel equalizer. As we would say, and it is changed or initialized in the training phase when training symbols are transmitted, here is a minor addition to this idea where you would also continue updating the filter coefficients or computing the optimal filter coefficients during transmission of useful symbols or useful bits or information. And the idea would be as follows from the base station now, rather than a training sequence you use, you transmit the symbol sequence, which is basically unknown of course to the mobile receiver. The mobile receiver uses its initialized equalizer to produce an output signal. And then you can use this output signal to make decisions about the transmitted bits or symbols of the output of the adaptive filter or optimal filter. That signal is used to decide whether with a high probability, let's say either a 1 or a 0 has been transmitted from the base station decided was a one with a high. Probability then you can use this one as if it were a transmitted training symbol and continue with the adaptation of the optimal or adaptive filter. This is referred to as decision directed operations. So the operation is directed by decisions made on the output signal from the optimal or adaptive filter.

16

With the examples from the previous slides, you should be convinced that the basic setup for an optimal or an adaptive filter is a valid one, or a useful one useful in whole collection of applications. For instance, if you dissect your smartphone, you will see two or three instances of optimal and adaptive filter. So it is a very useful set up and so this is where our study truly starts. And in this chapter we will look into optimal filter design, Wiener filter design in the next chapter we'll move from optimal filtering into adaptive filtering. So optimal filtering first and the basic setup is repeated here. So we're designing a filter with filter parameters filter coefficients, which is such that when the filter is fed with an input signal and again this, the signal will be or off the signal, we will have a prioritized statistical information.

And we'll see later what that actually means. So the filters fed with this given filter input signal produces a filter output signal, and the idea is that the output signal is optimally close to the 2nd signal, which is the desired signal, so that the error signal the difference between the filter outputs and the desired signal is optimally small. When we continue, basically we have to answer 2 basic questions. First question is if we're designing a filter, what type of filter, what type of filter

structure are we actually going to be using? And the answer to that question will be given in the next slide. And the answer will be that we're going to look into FIR filters only. This is a pragmatic choice. Because FIR filter design optimal FIR filter design leads to simple mathematics, etcetera. That's what we would like to see. So more on that in in the next slides.

The second question that we have to answer is when we say the error signal has to be small, so the filter output signal has to be optimally close to the desired signal. Obviously we have to define the cost function or at a distance measure. And here again, there's many choices, but we'll make a pragmatic choice again in terms of a quadratic B square. So you could say cost function, which again leads to simple mathematics as you will see in one of the later slides.

17

So the first question is what type of filter structure or filter we are going to be using it and the answer is the simple one. We're going to stick to basically FIR filters. FIR filters are finite impulse response filters, also referred to as tapped delay line filters. When you look at the structure you see a delay line which is tapped and then into a weird combination etcetera and another name for FIR filters is transversal. Filter again sort of inspired by the block scheme. Anyhow, we're going to use FIR filters and the block scheme is repeated here and FIR filter has an input signal u of k with K and square brackets sometimes or oftentimes we're going to use shorthand notation where U with the K and square brackets is actually replaced by a U with a subscript a K, and similarly for other quantities.

The UK is fed into a delay line, so the delay line produces UK UK -1 UK -2 up two General UK minus L for an L order filter. And then you do a linear combination of the basically the outputs from this delay line. So you do a linear combination of UK UK –1 up to UK minus L and the weights of the that you use in the linear combination are denoted here by the W0W1W2 up to WL for an L order filter in optimal and adaptive. The filtering theory we usually indeed use W for the weights for the filter coefficients, whereas in the previous chapter for an FIR filter, the weights would have been denoted by means of the B's. The B coefficients. The B's are exactly the same as the W's. Here merely sort of to irritate you, we have a bit of a change of notation here, but just remember, the W's are basically the B's so that the filter coefficient.

So, the output of the FIR filter Y at time K is basically and that is indicated in the first formula is basically a convolution operation of the input samples with the samples of the filter impulse response. The filter coefficients that W and alternatively, you can also spell this out as an inner product of two vectors where one vector that polyphase W vector has all the filter coefficients from W 0 to WL for an L for the filter, and then the polyphase U vector at time K has all the input samples U at time K, U at time K - 1 up to U at time K minus L. If everything is real valued as we assume it's here, the output signal can indeed be computed by means of an inner product: the polyphase W vector has transposed the polyphase U Victor, or vice versa. The polyphase U victor transpose times the polyphase W vector, so all this everything on this slide is basically meant to just set notation, remember that U is the input signal to the FIR filter. The W's are the filter coefficients. Replacing the P coefficients from the previous chapter. The output signal of the FIR filter is Y, and then two additional notations in a sense here. D of K is the desired output signal of the optimal filter, and then the difference between the desired outputs D and the actual output Y is referred to here as the error signal, which is indicated as E at time K.

18

Next to optimal FIR filters, of course you could think of optimal IIR filters, infinite impulse response filters. You could even think of more general nonlinear filters. But all this is not trivial and something that we're not going into, so we'll stick to FIR filters, which leads to simple mathematics. So that is basically a pragmatic choice, and we'll stick to it in the rest of the chapters.

19

Minor comments here. We're looking into optimal FIR filter design. You can also easily generalize this to so-called multi-channel FIR filter design where you have multiple filters with multiple input signals. Actually, we've seen an example of this. If you go back to slides 11 Where we looked into acoustic noise cancellation, they had a reference microphone

signal producing the desired output signal of the optimal filters. But then you had several noise reference microphone signals, and then each reference. Or noise reference microphone signal could be filtered by its own filter with its own filter coefficients, and then all these filter coefficients together would be jointly adapted until you have an optimal setting, and this is illustrated in this slide. So here you have it a three channel.

Example, you have first, second and third input signal, the first input signal gets filtered by in this example, the 3rd order FIR filter with coefficients W 0 up to W3 and then the 2nd signal gets filtered by another FIR filter with filter coefficients W4 up to W7. The order of this filter could even be different from the order of the previous filter, etc. Similarly for the third input signal, the outputs from the three filters are then added and this is compared to the desired signal and so it's the error signal is basically the difference between the desired signal and then the sum of the outputs of all the three in this case filters and this error and an adaptive filter set up for instance in the next chapter will steer the adaptation for all the filter coefficients and all three filters together.

20

And then further to this special case of the multi-channel filter is actually something which can refer to as a linear combiner or basically in the previous slides all filters are zero order filters with only one filter coefficient. So, if the order of all the filters in the previous slide is 0. The one coefficient basically the overall filter output would be W of 0 times the first input signal plus W1 times the 2nd signal plus W2 times the 3rd signal. So, you're doing a linear combination of the filters and search for the optimal linear combination to optimally approximate the desired output signal. So, this is referred to as a linear combiner. You can view it as a special case of the multi-channel FIR filter, but then again.

You could view an FIR filter as a special case of the linear combiner, where the different input signals are actually related to each other as where one is basically a delayed version of the other one. So, if all input signals or delayed versions of basically the first input signal. Then that indeed leads to an FIR filter. So, we started from a FIR filter. You can generalize it to a multi-channel FIR filter. Then consider a special case, the linear combiner. As you see it here and then again, a special case of the linear combiner is.

In fact, an FIR filter back to where we started from. So, everything we're going to see in later slides is easily generalized or tuned towards the multi-channel case or the linear Combiner case.

21

Second question that we have to answer concerns that distance criterion or optimization criterion when we say that the filter output has to approximate the desired output signal d of k optimally, or equivalently that the error signal the difference between these two signals has to be small. What do we actually mean by small? How do we quantify small and then optimize this?

Here again, we're going to use a pragmatic choice which leads to simple mathematics and pragmatic choice is to use the so-called and famous minimum mean square error criteria where we're minimizing the mean of the square of the of the error as you see it in the formula here is the expected value operator the mean operator.

So the constraints you're in is the MMSE criterion. The mean squared error criterion where we take the mean of the square of the error sample at time K and the error sample at time K is dK minus Y of K and the Y of K is defined in the previous slides to be or can be expressed as the inner product of the polyphase U victor with the polyphase W Victor. The W has all the filter coefficients and the U at time K polyphase U at time K vector has to last so many input samples at time K. (the formula in slide)

22

The MMSE criterion of the cost function of the previous slide can immediately be expanded into an alternative expression. As you see in this slide, the final formula here is the MMSE criterion split into three terms. The first term is a constant term.

It will not play a role in later slides.

The second term is a quadratic term and the filter vector the polyphase U, and the third term is a linear term and in the polyphase vector W. The quadratic term which you see appearing as is a matrix which is denoted here as a XUU. And we will identify this as the correlation matrix of the input signal, so this is the expected value of UU transpose.

And more details on this in the next slice. Whereas in the linear term you see a vector appearing which is expected value of U * d which is cross correlation between the input signal together with the desired output signal. So, in this expression you see a Matrix appearing which will be your correlation that matrix and you see a vector appearing which will be our cross-correlation vector. More on the cross correlation or I should say on the auto correlation matrix in the next slide.

23

The correlation matrix for the input signal U has a special structure and this is made a bit more explicit here in this slide, if you immediately go to the last formula or correlation matrix Xu was defined to be the expected value of UU transpose U polyphase U vector is a column vector. After let's say, and you multiply it with U transposes as a row vector. So column vector times row vector is indeed a matrix. Now then, if you look into the individual entries of this matrix, look into the one one element which is colored here. The 11 element is basically expected value of U at time K the K sample of the input signal times U at time K so expected value of U K ^2 basically.

This is defined to be the autocorrelation coefficient with zero lag of the input signal, and it's defined in the first formula XUU for lag D is the expected value of UK times UK minus delta for a lag delta, we're considering or we're assuming that the UK is basically a realization of a stochastic process which is stationary and then indeed you can define the sequence of autocorrelation coefficients like this where the autocorrelation is only a function of lag delta and not of the time index that you see appearing in the definition, which is the K. So returning to the last formula the 11 entry in that correlation matrix is expected value of UK times UK, and that is indeed the autocorrelation coefficient for lag delta is equal to 0.

If you then look for instance in the two element which is also colored in the in the matrix which you would see there is expected value of UK -1 times UK -1, but that again corresponds under the assumed stationarity of the stochastic process. That again corresponds to the zero lag. Autocorrelation coefficients are the same auto correlation coefficient, and similarly for the three elements in the matrix, etcetera. All the colored entries on the diagonal basically are the zero lag auto correlation coefficient.

It's similarly if you look into the off-diagonal elements in the matrix. So for instance, look into the 21 element, 2nd row, first column that would be expected value of UK -1 times UK, which corresponds indeed to the auto correlation coefficient where the lag is equal to one and same thing for the 32 elements and the third row, second column and the and same thing for the 4 3 elements, four 4th row, third column, etcetera. So if you again run down this diagonal, you would always see the lag is equal to 1 autocorrelation coefficient. And if you continue feeling that matrix, you indeed see that the matrix takes a special form or has a special structure which is referred to as a Toeplitz structure, where you always see the same number or entry when you run down the diagonal. So for instance, the main diagonal, you always have to 0 lag autocorrelation coefficient in the diagonal below the main diagonal you always have the autocorrelation coefficient lag equals to 1, and then the next diagonal autocorrelation coefficients lag equal to 2, etcetera. This is referred to as a Toeplitz matrix. It's also symmetric, so the 2-1 element is equal to the 1-2 elements. Basically because of the auto correlation coefficient expected value of UK times UK -1 is equal to expected value of UK -1 times UK etcetera. So the matrix takes a special form structure doped structure. It's symmetric can also be proven to be non negative definite.

24

Returning to the MMSE cost function where we had the quadratic term with the correlation matrix and then the linear term and the filter vector, the polyphase U with the linear term has the cross-correlation vector where you can easily identify the individual entries of this cross correlation vector, which is basically an expected value of some sample to U

signal times the D at time K, so that is the definition of your cross-correlation vector. So, you have quadratic term with the matrix Xuu linear term with the vector Xdu this is a simple quadratic cost function, convex cost function with in the general case a unique minimum which can obtain by setting the gradient equal to 0. Simple operation that eventually or immediately leads to set of equations and the polyphase W vector filter vector, which is referred to as the Wiener Hopf equation. So in the lower left corner of the slide you see the resulting equation from which you can compute the optimal filter coefficients. The Wiener filter coefficients. Hence the subscript WF for Wiener filter and what you observe is that you can compute the optimal filter coefficients by solving this set of equations which is defined by our matrix to Xu.

The relation matrix together with our vector Xdu cross correlation vector. So this is a simple set of equations that you can solve. So closed form expression for the Wiener filter coefficients would be Xuu inverted times the Xdu this is a closed form expression in practice you would never solve a set of equations like this by inverting a matrix. You have better procedures to do this. Think of cost elimination and whatever procedure to solve a set of equations. So basically the result is extremely simple, so if you're designing an L order filter, let's say L is equal to 100, so the number of filter coefficients is going to be 101. The end result here is a set of equations. It's going to be 101 equations and 101 unknowns and this is the meaning of the Weiner Hopf equations. So you solve this set of equations and that leads to your Wiener filter solution optimal filter solution. Here again, you could say at the beginning of the chapter I said we're going to view signals as realizations of stochastic processes of which we have a prioritized statistical information available. Now the question was what type of statistical information a priori statistical available statistical information which you need to compute your Optimal Wiener filter and the answer here is that basically you have to Xuu and Xdu so that is auto correlation information of the input signal. Xuu, right? auto correlation coefficients that define the correlation matrix Xuu and then also cross correlation information. Cross correlation coefficients that defined cross correlation vector X two u with that a prior statistical information you can put up the in your Hopf equations and then solve the Wiener Hopf equations for the optimal set of filter coefficients.

25

The MMSE cost function that we're minimizing here in the previous slide can actually easily be visualized as well. For this, you can turn the MMSE cost function into an alternative form that can easily be verified that the MMSE cost function is indeed equal to the MMSE cost function evaluated in the optimal Wiener filter solution and then plus a quadratic term which is basically defined by the distance of a particular filter vector W from this optimal filter in the Wiener filter and then a quadratic term which is defined by the Xuu autocorrelation matrix or correlation matrix, and that is visualized here for simple two-dimensional case. So where the W vector has two filter coefficients and then you see the MMSE function plots here with the minimum for the Wiener filter solution and then a quadratic error performance surface that you can usually visualize this the shape of the air performance surface will then be fully defined by the Xuu and this is something we're going to use in in the next chapter. It will actually be the eigenvalue decomposition with the eigenvectors and the eigenvalues of the Xuu that define how steep this server performance the service will be in a particular direction.

26

Property of the optimal Wiener filter is the so-called orthogonality principle, which says that the error signal obtained when using the optimal filter is orthogonal to the input signals that are used for the estimation. So basically the signals that sit in the polyphase UK vector that is expressed in the first formula here, orthogonality basically is defined here to be the expected value or by means of the expected value of UK times the Ek, the Ek being realized when the filter is indeed the optimal filter. So, orthogonality is here basically equivalent sort of in statistical terms, is equivalent to uncorrelatedness, so the error signal is uncorrelated equivalently orthogonal to the input signals to the adaptive filter, and that is easily proven. As you can see it here in the formula where the end result is indeed equal to zero as a byproduct of this, you can also make a linear combination of the of the UK signals. And basically the one that leads to the output signal and then to find that or conclude that the output signal is also orthogonal onto the error signal. So, the output signal Y of K which

is in fact a linear combination of the signals in the polyphase UK vectors and also orthogonal or uncorrelated with the error signal. So, you could view the optimal filter as taking a desired output signal. That EK and then this is split into a YK output signal and an Ek resulting error signal where the two are basically orthogonal or uncorrelated.

27

Final remark here about solving the Wiener Hopf equations in the previous slide we saw that to compute the Wiener filter solution you have to solve a set of linear equations which was referred to as the winner of equations. So, this is basically L +1equations and L + 1 unknowns. If you use a general procedure like Gauss elimination or whatever to solve the set linear equations L + 1 equations in L + 1 unknowns, the computational complexity which would basically be L to the power 3 which is a large computational complexity.

Now the good news here is that the matrix that appears in the Wiener Hopf figurations is this correlation matrix which has the special structure. As we have seen it in the Toeplitz structure. And in the Toeplitz matrix basically all entries or all numbers in this matrix are defined by basically the first row or the first column of this matrix. So it's a matrix with a special structure and this special structure can actually be exploited to reduce the complexity of solving a set of linear equations based on this matrix, and so there are a number of algorithms available that exploit the structure to reduce computational complexity from order L to the three to order L to the two, which can be a significant reduction in computation complexity of famous algorithm. Here is the Levinson Durbin algorithm which is used very often in specific applications like speech, code etc. The details of this algorithm are provided in the next few slides, but this is something that we're going to skip here. For those interested, you can look into the slides, but we're never going to ask questions there.

33

The last few slides I have a bit of background theory on optimal filter design Weiner filter design and consider something which is referred to as the Unrealizable Wiener filter. In previous slides we have designed an FIR filter where the chosen filter order, I should say is the capital L. And then an interesting question is what happens when you change the L when you increase the L ultimately set L2 an infinitely large number. When we considered an L order filter, we sort of assumed realizability where we say we have a finite length FIR filter and it's causal. The number of filter taps is L + 1, so that is realizable in real time. Now if we remove all these restrictions of realizability and we say, let's design an infinitely long FIR filter with infinitely many filter coefficients and this is indicated here with the subscript infinity, so the filter will have infinitely many coefficients for positive K as well as negative K and negative K basically says that the filter does not even have to be causal.

So, this is not realizable and hence it's referred to as the unrealizable Wiener filter. The interesting question now is if you do this, what type of filter do you get and the idea is basically if you set the filter order to a finite number capital L. Perhaps somehow? Achieve an approximation to this unrealizable sort of ideal Wiener filter, which can be infinitely long. The design of the Wiener filter involves or relies on the Wiener Hopf equations, which is a set of equations and one equation of the set of equations, is specified here for 1K and then and then K can go from minus Infinity to plus Infinity to consider all the equations in the Wiener Hopf equations. This is perhaps a bit more clearly displayed in the next slide where these equations are put together into a large set of equations.

34

The Wiener Hopf equations can also be expressed something like this as an infinitely large set of linear equations with infinitely many equations and infinitely many unknown filter coefficients to W is running from W at minus Infinity up to W at plus Infinity. So infinitely many equations and infinitely many unknowns from here. You can also pick a sub matrix from the infinitely large matrix and a sub vector from the infinitely large right hand side vector as well as the vector of unknowns and with these basically you form the finite Nth order or L order you could also say, Weiner Hopf equations as we have seen them in previous slides.

35

In this infinitely large set of equations, in these Wiener Hopf equations, you can basically identify or recognize in the left hand side of the equation. So basically the matrix vector product operation convolution of on the one hand the sequence of autocorrelation coefficients of the input signal together with the sequence of filter coefficients, and that immediately can be expressed alternatively in the Z domain. So if you do a Z transform, you have an alternative representation of this infinitely large set of equations that Wiener Hopf equations, which is expressed here Xu of Z times the unrealizable Wiener filter, also Z transform thereof is equal to Xdu of Z where Xu of Z is the is defined to be the power spectrum of the input signal UK or the input stochastic process u defined as the Z transform of the sequence of autocorrelation coefficients, and similarly the XDU of Z is the cross power spectrum between D&U, so the Z transform of the sequence of cross correlation coefficients.

So you have a very simple Z transfer or Z domain expression or version of the Wiener Hopf equation and then obviously the solution of this is provided here is the last formula. It's basically the cross power spectrum of the D&U divided by the power spectrum of the input stochastic process U and this is an interesting formula, so It's a sort of transfer function version of everything we've seen up till now. If you compare that to the Wiener filter solution, I'm on slide 24 where the Wiener filter solution was expressed as an inverse of a matrix times a vector and Xu matrix times an XDU vector. Here you see the sort of Z domain or Z transform or transfer function version of exactly the same thing basically.

36

With the expression for the unrealizable Wiener filter in the previous slides. You can derive and we're not going to see the derivation here. You can derive the two formulas which are provided in this slide. The first formula basically says what the MSE is that is achieved by the Unrealizable Wiener filter and in the sensor unrealizable Wiener filters, then sort of the best possible Wiener filter. So that gives you a mean squared error which is going to be small and any other Wiener filter will always have an MSE which is larger. Hence this MSE is also referred to as the irreducible error and an expression for this irreducible error is given by the 1st formula here. Ignore the formula as a whole that is not overly important.

But try and understand the concept of the irreducible error, so that is the MSE achieved by the unrealizable Wiener filter. Then you could ask where will be the MSE achieved if a fixed order of the optimal filter to be L. So for an L order filter what MSE you achieve and that is given by the 2nd formula. Again, the derivation is not provided here, but here the formula is basically interesting. It says that the MSE achieved by the L order filter is the MSE achieved by the Unrealizable Wiener filter which is the irreducible error and then plus an additional term which will always be positive. So the MSE for W is always larger than the MSE for the unrealizable inner filter and the additional term is interesting as it sort of refers to or it reminds us of FIR filter design.

Yes, we're going to integrate over all regional frequencies through the weighted version between the desired filter, which is the unrealizable Wiener filter and then the Lth order designed filter that we're trying to put up here, which is the W evaluated for radial Frequency Omega. So we're looking at the difference between Lth order filter and the infinite order filter. The Unrealizable Wiener filter we're evaluating the difference for each radio frequency and absolute value and then square.

And this is weighted by a weighting function. Remember weighted least squares filter design from Chapter 4, where the weighting function is basically the power spectrum of the input signal. So this is our weighting function. And then this weighted distance function or error function is integrated over all radial frequency. So to the Long story short, you could say that the MSE realized by the Lth order filter is basically the irreducible error plus an additional term that says how well you can approximate the unrealizable Wiener filter by means when Lth order filter, when you basically do FIR filter design.

37

The first example here to illustrate this and this is the acoustic echo cancellation example. So remember a loudspeaker plays the far end signal which is denoted here is the U of K.

And this loudspeaker signal couples into the microphone. We're assuming here that the acoustic propagation can be modeled by means of a transfer function. Agency from the loudspeaker into the microphone. So the microphone or the contribution and the microphone signal of the loudspeaker signal is basically the U signal filtered by the H of Z, where the H of Z is obviously an unknown transfer function, and we're assuming that everything behaves linearly, etcetera and can be modeled by means of a linear transfer function.

The second assumption that we make is that the microphone signal also may contain a near end speech signal, which is denoted here as n of K. And we're assuming that this n of K is uncorrelated with the U of K. So we're making an independence assumption expected value of N of K. Multiply it with any sample of the U signal is equal to 0. If you use that and you plug everything into the formula for the unrealizable Wiener filter that we have derived a few slides back, you can do the derivation and the end result is that the unrealizable Wiener filter is H of Z. So the best possible setting for the ECHO cancellation filter is H of Z. Obviously H of Z. In practice everything indeed behaves in the linear fashion. Etcetera.The acoustic modeling is indeed a linear transfer function H of Z. This in practice will be causal, and if it has a finite order, then indeed it's actually a realizable Weiner filter in this case.

38

Further to this. If you compute the irreducible error so the MSE for the unrealizable or realizable, whichever way you want it, filter the H of Z. If you plug that into the formula for the irreducible error. Here we're going to skip all the details of the derivation, but the end result is the expected value of N K ^2. So it's basically the power of the near end signal and that also makes a lot of sense if the filter that you use in the Echo canceler is H of Z. Then the error signal the residual error signal is basically the N of K and so the irreducible error is indeed the power in the near end signal and that also makes sense. The idea of the Echo canceller is to cancel the echo and not so much to cancel the near end signal of course.

39

Second example to illustrate this is the channel equalization example. So you'll transmit this signal from a base station to a mobile receiver. The radio channel is assumed to be or we assume that the radio channel can be modeled by linear transfer function H of Z and then the received signal also has a noise contribution which is indicated here as N of K. So overall, the received signal in the mobile receiver, the U of Z which will be inputs to the optimal filter is basically H of Z times the transmitted to signal transmitted from the base station to D of z plus the noise term which is N of z. Again, we're making an obvious independence assumption in that the transmitted sequence and the noise contribution are basically uncorrelated.

40

With this you can drive an expression for the unrealizable Wiener filter again, the expression is given here. It's a fairly simple derivation. The expression as such perhaps doesn't ring a bell, but you can consider special cases. For instance, in the case where the noise signal is 0, there's no noise contribution. Then it turns out that the unreal, Unrealizable Wiener filter is actually equal to 1 / H of Z, so the inverse of the radio channel model. And that makes sense. So if there's no noise, the best you can do is basically invert the channel transfer function to sort of ideally reconstruct the transmitted bits or symbols. Now in the case where there is actually a noise contribution, just or merely inverting the channel may not be a good idea, because the 1 / H of Z also multiplies the noise contribution, and that could actually lead to something that is referred to as noise amplification and could lead to a lot of decision errors eventually and hence could be a bad idea. So, if there is indeed a noise contribution. The 1 / H of Z is not the sort of the best possible unrealizable Wiener filter. If the noise actually goes to Infinity. If the noise contribution is infinitely large, the best thing you can do is actually set the filter to 0, because there's no sort of identifiable recognizable desired signal anymore in the received signal it's nothing but noise

basically and so hence the unrealizable Wiener filter converges to 0 for the case where the noise becomes infinitely large. With this you can do the formula for the irreducible error but keep the formula at that perhaps doesn't at all ring a bell here.

5 Chapter 8

Chapter 8

5.1 Slide 3:

In this slide and in the next four slides, we have a brief review of what was covered in the previous chapter on optimal filter theory. So, the basic setup for optimal filtering is repeated in this slide. Remember that signals are viewed as realizations of stochastic processes, of which we have statistical information available. We'll see in the next few slides what type of statistical information we need to do optimal filter design. And the setup is - where we we're designing a filter that is fit with a filter input, given signal filter input, and filter produces an output where the idea is that the filter output is optimally close to the desired output signals, so that the error signal, which is the difference between the desired signal and the actually produced filter output, is as small as possible.

5.2 Slide 4:

The filter is always going to be an FIR (finite impulse response) or tapped-delay line filter. We will not look into IIR filters and more general filter structures. And so, this slide has a bit of notation that we will use in the rest of the chapter. The input signal to the filter is known as u[k], where k can be between square brackets or can also be used as a subscript. So, input signal is u of k and then the filter coefficients are w's, which in the previous chapter are the b coefficients of a difference equation of a finite impulse response filter. But here they are referred to as w's for the weights. So, the weights w from w0 up to wL for an L order filter. And so you delay the input signal, make a linear combination of the delayed input samples to produce an output signal which is known here y of k (y[k]). Then the y[k] is compared to the desired output signal, which is the d[k]. The difference between d[k] and output signal y[k] is the error signal e[k]. Finally, the description of the filtering operation, the filter it produces or executes a convolution operation. Obviously, if the input signal with the filter coefficients the w, this is expressed by means of the summation operation in the formula here, which can also express this in a simpler form by means of vectors. A bold w vector with all the filter coefficients, and a bold u at time k vector that collects the input samples from u at time k. Up to u at time k-L for an L order filter and then the output signal is basically the inner product of the two vectors - either w times the u vector or the other way around the u vector times the w vector.

5.3 Slide 5:

Next to a filter structure, we also need a cost function, which is basically a distance measure to measure the distance between the desired signal d[k] and the filter output signal y[k]. For this we use the MMSE criterion or the Minimum Mean Squared Error criteria, which is basically where we specify the mean of a square of the error. So, we're considering the mean, which is the notion here by the expected value operator, the E operator of the square of the error signal. So, we're computing the error signal as dk - yk, and then the squared error, and then the expected value of the square. That is the

Minimum Mean Square Error criterion where the y signal can again be expressed in terms of the inner product of the bold u and the bold w vector.

5.4 Slide 6:

When you expand the MMSE cost function, you see a matrix appearing and a vector appearing. The matrix is denoted here as Xuu. And the vector to know here is Xdu. One is an auto-correlation or a correlation matrix of the input signal, and the vector is a cross-correlation vector between the input signal and the desired output signal. It turns out that if you then minimize the cost function by setting the gradient equal to zero, you end up with a set of equations and the optimal set of filter coefficients and that set of equations is referred to as the Wiener-Hopf set of equations and that is indicated to the bottom left of the slide. Xuu times the optimal set of Wiener filter coefficients w, is equal to Xdu, the cross-correlation vector. So, in a closed form formula and the Wiener filter solution, optimal filter solution would be the inverse of the auto-correlation matrix times the cross-correlation vector. Even though you would never actually compute a matrix inverse to solve a set of equations, we have better ways. To do this, the important part here is that on the one hand you see that optimal filter solution results from solving a set of linear equations, which is very simple. So, if you have w vector with 100 coefficients, basically you will have 100 equations in 100 coefficients. So conceptually that is very simple. And the other thing to observe is that what you need in order to solve or to compute this optimal set of filter coefficients is basically the auto-correlation coefficients of the input signal that defined the Xuu matrix, which by the way has a very special structure, namely the Toeplitz structure that we have seen in the previous slide. But so you need the auto-correlation sequence of the input signal to construct the Xuu and then also you need a cross-correlation sequence between the u and the d to construct the cross-correlation vector Xdu. So basically you need second-order statistical information to optimize a quadratic statistical or stochastic cost function. That all makes sense. And so, this is the statistical information that you have to have available about the input signal and the desired output signal to compute the optimal filter coefficients.

5.5 Slide 7:

The next question is how do you apply such optimal filter theory in a practical setting. If we say that in order to design an optimal filter we need auto-correlation information or coefficients of the input signal and cross-correlation information between the input signal and the desired output signal. Think of a practical application like echo cancellation or whatever something we have seen in the previous slide. In practice you will never be given this set of auto-correlation and cross-correlation coefficients. You would have to estimate these from observed signals/samples and even these auto-correlation and cross-correlation coefficients could vary over time. So, this is where we're going to need a different version of the same basic approach of optimal filtering, and a version which is referred to as adaptive filtering, where the system adapts itself to the signal that it observes. And so it basically has or will have two processes. On the one hand, the adaptive filter will be a filter or act as a filter because eventually we're designing a filter which is meant to filter an input signal to produce an output signal. But on top of that we will have an adaptation process where basically the filter looks into the signals that it processes/produces, and adjusts its coefficients, and it will turn out that this adaptation mechanism basically corresponds to some sort of feedback operation where the error signal is fed back to steer the adaptation of the filter coefficients, which indeed makes sense. If the error signal is small or zero for instance, then you should be happy with the current filter and, and shouldn't do any updating of the filter coefficients. Whereas if the error signal is large, then probably you need to update your filter coefficients because that filter is not a very good filter for you.

5.6 Slide 8:

The first adaptive filter system or algorithm that we're going to study is the famous LMS or Least Mean Squares algorithm. This is an incredibly simple algorithm as you will see, a bit embarrassingly simple I would say, but at the same time it's an incredibly popular algorithm which is used in many applications. As I would say, if you would dissect your smartphone, there's a high probability that it will have at least one, maybe several instances of the LMS algorithm.

5.7 Slide 9

The starting point for the derivation of LMS algorithm is Wiener filter design, optimal filter design. In the previous chapter and in the review in this chapter, we've seen that you can compute the Wiener filter by solving the Wiener-Hopf equations which result from setting the gradient of the MMSE cost function to zero, and that is basically L+1 equations in L+1 unknowns. And you have procedures to solve sets of linear equations like the standard cost-elimination-based procedure or because of the fact as we have seen it in the previous chapter that Xuu is a structured matrix - Toeplitz matrix. You can do things like the Levinson Durban algorithm etc. Now an alternative way of computing the Wiener filter is to optimize/minimize the MMSE criterion and by means of an iterative procedure. And the basic iterative procedure to do this is the well-known steepest-descent iterations method. And in this method, you have an iterative procedure. You compute in a sense the next estimate for the w vector from a previous estimate. So that's what you see in the formula here where n is the iteration index. So, w(n+1) iteration is the previous w(n) nth iteration plus a correction to that. And the correction is basically a step in the direction of the gradient vector which is obtained by computing the first-order derivative of the cost function. And evaluating that for w is equal to the current iteration w. And if you compute that gradient, you end up with obviously a very simple formula that says w(n+1) iteration is equal to w(n) plus a correction, which is mu times, and the mu is the step size parameter, which will have to tune and we will see more of that in the next slides. And then the expression for the gradient, which is Xdu-Xuu·w. When you go back to the Wiener-Hopf equations, you see that the optimal Wiener filter indeed sets this gradient equal to zero. So, if you would land after a number of iterations in the optimal filter, the Wiener filter, then this last term would be zero. And the update equations would say the next iteration is basically equal to the current iteration. So you stay in your optimal filter. You have reached the solution of the minimization problem.

5.8 Slide 10:

The steepest-descent procedure is a very simple procedure, but an important aspect is the proper tuning of the step size, the mu factor that you see in the basic equation. So the question is what is a proper setting for the mu and it will turn out there's indeed an upper and a lower bound for the mu. Derivation of this upper and lower bound is very simple and it's indicated in this slide and the update equation as you see it repeated here from the previous slide can immediately be turned into an alternative or equivalent form, which is the formula in the middle of the slide. And that formula basically says if you evaluate the error, you can define the error as the n plus first iteration, the error being the difference between the n plus first iteration and the optimal filter/Wiener filter, and then this error is equal to the error at the previous iteration at the nth iteration after a multiplication. Pre-multiplication with the matrix, and the matrix is identity matrix minus mu times the Xuu and so every iteration this error vector basically gets multiplied by this this matrix. So if you start from the initial value w at iteration 0 and an initial vector w(0) minus the optimal filter. Then after n plus first interations this initial vector gets multiplied by the same matrix over and over again and hence to the power n+1. Now the hope is of course that this error vector gets reduced to zero, so converges to zero. And the question is, what is a good step size parameters such that indeed the multiplication with this matrix leads to an error vector which is reduced or being reduced in each iteration. Now this actually depends on the eigenvalues of that matrix, the I-mu·Xuu, and it's the eigenvalues that define

whether this error vector will be reduced in each iteration, yes or no. The basic statement will be that the error vector will be reduced if and only if all of the eigenvalues of that matrix are stable, which means that in absolute value, they should be smaller than one, so they should basically lie between -1 and +1, assuming that they are real-valued. Now the eigenvalues of that matrix I-mu·Xuu actually are easily computed. It turns out that the eigenvalues are actually equal to one minus mu times the eigenvalues of the Xuu matrix. So, if you have the eigenvalues of the Xuu matrix and remember this is the matrix with a special structure. It's a symmetric and Toeplitz matrix. And because of the fact that this is symmetric matrix, you know that the eigenvalues are going to be real-valued. This is something to remember. So you compute the eigenvalues of the Xuu, these are the i's. And you know that these are real-valued and then you know that the eigenvalues of this other matrix I-mu·Xuu are basically one minus mu times the eigenvalues of the Xuu and with the i being real-valued and obviously also the step size parameter being real-valued. This whole expression is real-valued. So, if we say this absolute value has to be smaller than one, then basically 1-mu·i has to lie between -1 and +1. That basically leads to an upper and lower bounds and equivalent expression for the mu, which is given in the last formula, which can readily be derived from the previous formula. And the end conclusion is that the mu has to be on the one hand positive and on the other hand smaller than 2 divided by max, which is the largest eigenvalue of the Xuu -input signal auto-correlation matrix. The meaning of this is, like this mu has to be positive. So you compute the steepest-descent direction and with mu positive you step towards in the direction of the steepest-descent direction. Basically, you make a forward step. And because mu is positive and you shouldn't be stepping backwards, so that is the meaning of the constraint that mu has to be positive. And then mu has to be smaller than 2 divided by max, you take a step towards the optimal solution, the Wiener filter solution, but if your step is too large, then then you're basically passing the optimal solution and you could be further away from the optimal solution in the next iteration. So that defines basically, or that explains the upper bound for the step size parameter. An important thing to note here is that it's the largest eigenvalue of the Xuu- the input signal auto-correlation matrix, that sets the step size upper bounds.

5.9 Slide 11:

Next question is how fast will this steepest descent algorithm actually converge. For a given step size parameter mu. And to analyze this so-called transient behavior, we start from a formula again from the previous slide that tells us what the evolution of the error vector truly is. The error vector after n+1 iterations is w(n+1) minus the optimal Wiener filter. And that is seen to be the initial vector with the initialization w(0) minus the Wiener filter vector. And in n+1 iteration, this initial error vector gets multiplied with this matrix I-mu·Xuu matrix, n+1 to the power. Now here in this formula we're going to exploit or substitute the eigenvalue decomposition. Of the auto-correlation matrix the Xuu matrix we have exploited this already in the previous slide. Here it's made more explicit Xuu the eigenvalue decomposition is Q times a matrix times Q transpose. Q defines the eigenvectors. It has the eigenvectors in the columns and matrix is a diagonal matrix that with the eigenvalues on the diagonal. Same eigenvalues obviously as we have used them in the previous slide. So matrix is diagonal, Q has the eigenvectors in its columns and for a symmetric matrix, as we had it already in the previous slide, the s are real-valued but then also the eigenvectors form an orthogonal set of basis vectors. So the column vectors in Q matrix form an orthogonal set of vectors, and hence Q transpose times Q is an identity matrix that is a basic property of symmetric matrices and a symmetric eigenvalue decomposition. Now if we exploit this given eigenvalue decomposition and substituted it into the first formula, we eventually end up with the second colored formula, which has an interpretation something like this: what we have now is the error vector after n+1 iterations, but pre-multiplied with Q transpose and this is equal to then the diagonal matrix which is defined by the eigenvalues basically, times the initial error vector with w(0) etc. again pre multiplied by Q transpose. The meaning of the multiplication with the Q transpose is that we're basically transforming the error vector to or expressing it based on a different set of basis vectors which are defined here as the columns of the Q matrix. So the eigenvectors of the Xuu matrix. So basically, we're looking at the same error vector from a different perspective after a projection onto the eigenvectors of the Xuu matrix and these projected

components of the error vector, the evolution of those is basically sort of expressed in that formula and it basically says the error components after projection onto the eigenvectors evolve in such a fashion that this vector after n+1 iterations is the initial error vector after projection etc multiplied with a matrix and plus one times multiplied. But now the matrix is a diagonal matrix. So on the diagonal you would see 1-mu·i, i are the eigenvalues of the Xuu auto-correlation matrix and that is an interesting formula. It means that each component of this projected error vector basically evolves individually and gets multiplied by 1-mu·i (the corresponding eigenvalue) and so evolves independently and hence this component of the projected error is referred to as a mode. And so the i-th mode corresponds to the i-th eigenvalue and evolves according to 1-mu·i to the power n, so it gets multiplied by 1-mu·i in each iteration. With this you see that if you have small eigenvalues, then the corresponding mode gets multiplied by 1-mu·i in each iteration. But for small , this 1-mu·i could be very close to one and that basically means that the error vector or the component of the corresponding component of the error vector gets multiplied by a number which is very close to one and hence converges very slowly.

5.10 Slide 12:

In the previous slide we concluded that the convergence for the i-th mode is defined by 1-mu·i, from there you see that you will have the slowest convergence for the mode corresponding to the smallest eigenvalue which is to know here is min. The slowest convergence for the mode that corresponds to min and this mode the convergence of this mode is defined by 1-mu·min. And if you exploit the upper bound for the step size as we have derived it two slides back you would see and that is the colored formula here that this 1-mu·min sits between 1 which is an upper bound, and then 1-2(min/max) which is a lower bound. Bad news would be if the lower bound is also very close to one because that would lead to very slow convergence and this actually happens when min is much smaller than max and this is referred to as having a large ‘eigenvalues spread’. So if the eigenvalues or if at the same time you have very small eigenvalues together with very large eigenvalues that is referred to as having a large eigenvalue spread. And so in that case min would be much smaller than max and hence that left hand side expression in the colored formula would be very close to one, which would imply very slow convergence for the slowest converging mode. So min being much smaller than max is bad news in terms of convergence. So convergence can be very slow if you have a large eigenvalue spread, this is a take home message, something to remember. And then the question could be when does that truly happen. In the case where for instance the u[k] signal, remember that we're looking at the eigenvalues of the Xuu. An example is when u[k] is for instance a white noise signal. Then you can easily check that the auto-correlation matrix is an identity matrix from which all the eigenvalues are the same and that is basically that the best possible scenario. So, so then the min is equal to max. You don't have any eigenvalue spread, so a white input signal would be an ideal input signal. On the other hand, if the input signal u[k] is very colored signal, then you would indeed observe that the min is much smaller than the max and that leads to very slow convergence. So the steepest-descent algorithm works well for white input signals and could converge very slowly for very colored input signals.

5.11 Slide 14:

The LMS algorithm is a simple algorithm and a popular algorithm. It's also quite old algorithm. It was invented in the 60s by Bernard Widrow and his co-author. Bernard Widrow is a professor at Stanford university in the US you can watch the video here to see Bernard Widrow, who is still alive, by the way, at work. And so, you can see the algorithm as well as inventor are still going strong.

5.12 Slide 15:

Let us now move from optimal filter design as we have seen it based on the steepest-descent algorithm in the previous slides, to adaptive filtering and derive a first adaptive filter algorithm, namely the famous and very simple LMS or Least Mean Squares algorithm. Starting point is the steepest-descent formula from the previous slide, where now we have for convenience replaced the n iteration index by n-1 basically. So the formula will read w(n), so the iteration, the nth iteration is equal to the n minus first iteration w(n-1), plus a correction which is mu times, and then the gradient vector where we have substituted the definitions for the cross-correlation vector and input signal correlation matrix. The cross correlation vector is the expect value u·d at time k, and the correlation matrix is expect value of u· u transpose at time k. Now to work this iterative procedure and an algorithm that basically works over time, we're going to do a simple substitution. We're going to replace the iteration index that we have here, the n, and replace it by the time index, which is k, and the meaning of that is basically that we're running a steepest-descent interative procedure where we do basically one iteration per time interval. So, the new formula here is w at time k, so the filter vector at time k is equal to the filter vector at timek-1 plus a correction. So, that correction term mu times … still obviously has the expect value operations and this is something we would like to get rid of. This means that you have auto-correlation and cross-correlation coefficients there. So these statistical information which will in practice not be given to you. So what you would have to do here is estimate these auto-correlation and cross-correlation coefficients from the observed signals or samples. So if you see expect value of uk·dk, the way we're going to compute this is by perhaps assuming stationarity and then average this expression over a time window. So over a time window we're going to observe uk·dk and then average uk·dk over this time window. And the obvious question then would be how long would a suitable time window be. Are we going to average over 10 samples, or 100 or 1000 time samples? Now the remarkable choice that is made here to derive the LMS algorithm is that the length of the time window will be equal to 1. 1 is a small number, but it leads to a very simple algorithm where basically expect value of uk·dk is replaced by an averaging over one observation, so basically uk·dk, which effectively means that you're dropping, removing the expect value operations from the formula in the middle, which leads to the last formula. So expect value of uk·dk is replaced by uk·dk and then similarly expect value of uk·uk transpose is merely replaced by uk·uk transpose, and if you reorder some of the terms or factors, you end up with the final expression, which is indicated here, and this is the famous LMS algorithm. So LMS has a very simple algorithm. It's a one-line algorithm and the basic formula basically says the filter vector at time k is the previous filter vector at time k-1 plus a correction term, the correction term is mu times. And then you see basically two ingredients, uk - which is a vector of input signals at time k, which gets multiplied by something which is defined as ‘a priori error’. It's dk minus uk transpose times w_LMS at time k-1, so you're basically applying the previous filter vector to the current input signal vector uk and you're comparing that to the output signal dk and this is referred to as the ‘a priori error’. Remember that we're designing a filter which should be such that if you filter the input signal with that filter, it should approximate the dk and what you're basically doing here, you're using previous filter at time k-1 to filter the current input signal uk and compare that to the desired current output signal dk and check if this is large or small and that is indeed the a priori error.

5.13 Slide 16:

Observe that this very simple LMS algorithm with the definition of the ‘a priori error’ actually fully matches with the prototype adaptive filter setup that we have to find earlier, where basically the error signal, the difference between the desired output signal and the actual filter output signal is fed back and used for the adaptation of the filter coefficients. So indeed, the ‘a priori error’ is the difference between the desired signal at time k and the filter outputs at time k, where we're using the filter coefficients from the previous time step, k-1. And if the ‘a priori error’ is for instance zero, then basically you should be happy with the current set of filter coefficients and not change it. And this is basically what the update formula then also does. If the ‘a priori error’ is zero, then w[k] is equal to w[k-1], you do not change it. Whereas if

the ‘a priori error’ is large, then you are probably not so happy with the current set of filter coefficients and you modify the filter coefficients w[k] is w[k-1] plus a non-zero correction term. And the vector in which you do the correction is the input signals vector, the bold uk which gets multiplied on the one hand by the step size parameter mu, on the other hand by the ‘a priori error’. So that is the simple operation of LMS algorithm.

5.14 Slide 17:

The LMS algorithm is a very simple algorithm. It's a one-line algorithm and also in terms of computational complexity, it's a very cheap algorithm. It's a vector algorithm. In a sense we have a new vector which is an old vector plus a correction vector. In the computation of the correction vector, you would have the inner product of the u and w vectors and then the multiplication of the correction vector u by the ‘a prior error’ altogether. If the size of the vectors is L+1, so each vector has L+1 coefficients, the complexities on the order of 2L multiplications per time update. So, per sampling period, the number of operations is on the order of L, the size of the vectors. Basically, that is something to remember.

5.15 Slide 18:

The LMS algorithm is also a simple algorithm that you can draw. You can draw a signal flow graph which is actually sort of realization similar to the filter realizations that we have seen in chapter 5, so you can draw a signal flow graph with delay elements and multipliers and adders that take an input signal and a desired output signal and produce an output signal with the filter adaptation etc, all included. You can easily check that the operation of the signal flow graph indeed corresponds to the one-line LMS algorithm.

5.16 Slide 20:

The derivation of the LMS algorithm was very simple, but at the same time quite ad hoc. Remember that we have started from the steepest-descent equation or update equation and then merely sort of removed the expect value operations and that leads to a simple algorithm. But the question is does that really work in practice? If I use the LMS algorithm in a concrete application like an echo cancellation application or whatever, does it, does it really work well? And so here you have to do an analysis of the LMS algorithm, which unfortunately turns out to be quite complicated. LMS is a simple algorithm with a complicated analysis, so we're not going to look into the details of such theoretical analysis of the LMS algorithm. What we'll do is only highlight two sort of main results from such an analysis and result number one is in this slide, it's about the expected behavior of the LMS algorithm. And then a second result is related to what is referred to as noisy gradients which will be explained in the in the next slides. So first expected behavior. The simple statement here is that the expected behavior for the LMS algorithm is the steepest-descent behavior. And the way you should interpret this statement is something as follows. Remember that signals in the optimal and then adaptive filtering context are viewed as realizations of stochastic process. So the input signal, there's stochastic process, and realizations of that stochastic process produced the input signal. And similarly for the desired output signal. Now you can run the LMS algorithm for one realization of the input signal and desired output signal. So, basically you start the LMS algorithm at time 0 from an initialization w at time 0, and then you use one realization of the input signal process and one realization of the desired output signal process. Run the LMS algorithm and that leads to a sequence of filter vectors. So, w[0] the initialized filter and the w[1] w[2] w[3] … And you can envisage sort of a trajectory in an L+1 dimensional space or L+1 is the dimension of these vectors. So, a trajectory in an L+1 directional space which is defined by LMS algorithm. And so this is one run of the LMS algorithm. Then you continue with a second such run of the LMS algorithm. Start, let's say from the same initialization, but use a different realization of the input signal process, stochastic process and a different realization of

the desired output process. Again run the LMS algorithm leads to a different trajectory from the same w[0], but then the next w[1] will be different and then w[2] etc. Different trajectory for the second run of the LMS algorithm and on you go you do a third run, a fourth run, etc, infinitely many runs that lead to infinitely many such trajectories and then you would basically average overall these trajectories and the outcome would be that the average trajectory basically corresponds to the steepest-descent trajectory. This is basically because in the LMS algorithm you have removed the expected value operations from the update equation. But then if you average overruns you're re-introducing the expected value operations and sort of you're back at the steepest-descent behavior. So, simple conclusion here, the expected behavior of the LMS algorithm is the steepest-descent behavior and hence the step size parameter that we use in the LMS algorithm should satisfy the same upper and lower bound as we have derived it for the steepest-descent algorithm. So the mu step size parameter should be positive and upper bounded by 2/max, the maximum eigenvalue of the input signal correlation matrix.

5.17 Slide 21:

The second result from the theoretical analysis of the LMS algorithm is somewhat more remarkable and it's related to something which is referred to as noisy gradients. Assume you apply the LMS algorithm and assume for once you're lucky and after a number of iterations you land in the optimal solution in the Wiener filter solution. So assume that at time k-1 the LMS vector at time k-1 is equal to the optimal filter the Wiener filter. Then when you do the next update of LMS algorithm, you use an estimated gradient which is a part of the update formula which is repeated here. It's k times the ‘a prior error’. Now if you compute the expect value of this estimated gradient so you add expected value for the first term uk·dk, and then an expect value for the second term minus expected value u transpose times filter vector. That is actually equivalent to the Wiener-Hopf of equations and then you would decide that the estimated gradient expect value would be zero. And that would be ideal because it would mean if you're lucky and you land in the optimal solution, then the computed gradient to zero and you stay in the optimal solution. But now we have dropped the expect value operations and so instantaneously estimating this gradient and instantaneously if you plug in actual samples as you observe them at time k for this expression, the computed gradient, the estimated gradient is never going to be zero. This is like two signals of which the cross-correlation is zero. So the expected value of one sample or sample of signal one times sample of signal two is zero. But if you take instantaneous samples and you multiply them together, a sample of signal one and a sample of signal two, the result is never going to be zero. So the observation is if the LMS algorithm, you're lucky and you land in the in the wiener filter solution, the steepest descent algorithm would keep you there in the optimal solution, but the LMS algorithm has a non-zero estimated gradient and then we'll do an update that moves basically away from the optimal solution. So you land in the optimal solution, you're lucky, but then in the next update you're already away again from the optimal solution.

5.18 Slide 22:

Conclusion is that the estimated gradients that are used in the LMS algorithm are not exact gradients, so they're referred to as noisy gradients. And you can envisage that LMS process or convergence process as the LMS algorithm. Assuming that the step size parameter is properly attuned according to the upper and the lower bound from the previous slide, then the LMS algorithm converges to the optimal solution but actually not exactly to the optimal solution, rather to something like the neighborhood of the optimal solution where the LMS algorithm keeps sort of moving around to the optimal solution in the neighborhood of the optimal solution. Now the optimal solution minimizes the mean squared error. So the LMS algorithm sort of moving in the neighborhood of the optimal solution will always correspond to a larger mean squared error. So we're minimizing the mean squared error, but the LMS algorithm, after infinitely many iterations will

still achieve a mean squared error which is larger than the minimum mean squared error which is achieved by the optimal solution and that effect is referred to as the excess MSE effect. The excess MSE is the extra MSE which is a result of this sort of movement of the LMS solutions in the neighborhood of the optimal solution. The MSE, which is realized by the LMS algorithm bomb convergence to the neighborhood of the optimal solution, let's say so after infinitely many iterations, is the minimal MSE which is realized in the optimal solution plus an excess term which is then referred to indeed as the excess MSE. And the excess MSE can be quantized and theoretical result here is that the excess MSE is basically equal to the minimum MSE realized for the optimal solution and then multiplied by a factor which is referred to as the mismatch. And a mismatch, here is two ingredients. On the one hand it's defined by the step size parameter, the larger the step size, of course the larger the neighborhood of the optimal solution in which the LMS algorithm sort of moves. So larger step size leads to larger mismatch and then larger excess MSE. And then the second ingredient turns out to be the sum of all the eigenvalues, those eigenvalues that we have used in previous slides, eigenvalues of the input signal auto-correlation matrix. Now, the sum of the eigenvalues there, there's a number of equivalent expressions which are provided in this slide. The sum of the eigenvalues of a matrix is also equal to the trace of the matrix, which is the sum of the diagonal elements of the matrix. And this matrix, the auto-correlation matrix Xuu, remember it's a Toeplitz matrix where all the diagonal elements are basically equal to each other, are merely the zero lag auto-correlation coefficient, so the trace here is basically L+1, which is the dimension of the auto-correlation matrix times the zero lag auto-correlation coefficient which is the expect value of uk squared. It says something about the power indeed of the input signal. So now with this excess MSE which is of course bad news. The idea is to minimize the MSE and so you will get extra MSE in order to minimize this excess MSE or the mismatch. A strategy would be to set the step size parameter to a smaller value or the upper bound of the step size parameter to a smaller value that bound the mismatch and hence the excess MSE and the strategy here is to say the excess MSE can be 10% of the minimal MSE. So the mismatch should be smaller by an upper bound which is derived here then and is equal to 0.2 divided by, and then an expression based on the trace operation. You can also use the expression with the summation of all the ’s. And eventually that means the upper bound for the step size parameter will be much, much smaller than the initial upper bounds derived from the analysis of the steepest-descent algorithm. So, the initial upper bound was mu was smaller by 2/max. Now step size parameter should be bounded by 0.2 instead of 2, so this is 10 times smaller. And you divide not by only max, but you divide by the sum of all the eigenvalues which is the max plus all of the other eigenvalues. So that is much larger than max and hence that further reduces the upper bound for the step size parameter. So, conclusion here is basically, we start, or we derived the LMS algorithm from the steepest-descent algorithm, steepest-descent algorithm can have slow convergence when you have a large eigenvalue spread, and the convergence is also dictated or defined by the step size parameter mu. And here it turns out that you're going to have to use the LMS algorithm, a step size parameter, which is even much at least 10 times smaller than the step size parameter in the steepest-descent algorithm, which will lead to, again, a much slower convergence for the LMS algorithm.

5.19 Slide 24:

The LMS algorithm is an extremely popular algorithm. It's an extremely simple algorithm, one-line algorithm, basically. People obviously like simple things. On the other hand, convergence of the LMS algorithm can be quite slow in cases that we have studied in the previous slide. So for instance, when you have a large eigenvalue spread corresponding to a very colored input signal. So, simple algorithm but not always an ideal algorithm and hence people have spent lifetimes to improve upon this basic algorithm and derive sort of variations on the same theme and that has led to a whole collection of LMS type algorithms. A popular variation is the Normalized LMS algorithm is s actually derivation of the LMS algorithm which is often used in practical applications and will study this and the next two slides, simple modification compared to the initial LMS algorithm. And then you have things like transform domain LMS, block LMS, where basically you try to further reduce the complexity of the LMS algorithm by keeping the LMS filter vector constant for a whole block of samples and do only one update of the filter vector per block of samples. And the advantage is basically that as you keep the filter

vector fixed for a whole block of samples, there's a filtering operation which can be cheaply computed in the frequency domain and that leads to frequency domain LMS. We're going to see further details in a later chapter, chapter 13. So ignore at this point and then there's for instance subband LMS algorithms and many variations on the same theme.

5.20 Slide 25:

The variation of the LMS algorithm which as I mentioned in the previous slide is often used in practice is the so-called normalized LMS algorithm. The basic formula is given here again it's a one-line algorithm. The difference basically is that the step size parameter the mu that you had in the LMS algorithm is now replaced by what is referred to as a normalized step size parameter. So you have the mu bar which is then eventually going to refer to as a normalized step size parameter which gets divided by a function of a characteristic of the input signal basically. So it's divided by u transpose u which in a sense measures the power of the input signal, plus a regularization parameter , which again should be tuned. So the only difference is in the step size parameter, the mu is replaced by mu bar divided by something which is input signal-dependent. The complexity of the normalized LMS algorithm with this is slightly increased. To compute the normalization factor you have to execute an additional vector product u transpose u, which requires another order L multiplication. So, unless for the case where you compute this recursively which can be done, unless you would do that, the complexity is the order of 3L rather than 2L for the original LMS algorithm.

5.21 Slide 26:

The main advantage of the normalized LMS algorithm is that it will have simple tuning rules for the normalized step size parameter, the mu bar. if you go back to the slides that explain to the noisy gradient effect, there we drive an upper bound for the initial step size parameter the mu where the upper bounce is a function of the basically power of the input signal. It had expected value of uk squared. Now here this expect value of uk squared is basically estimated by some sort of averaging operation in this vector product u transpose u that you see in the normalization of the step size parameter. So, that aspect is sort of built in here and hence that the mu bar that remains indeed has very simple upper and lower bound. The conclusion here from a normalized LMS analysis would be that you're going to have convergence in the steepest-descent meaning when the mu bar is positive and then upper bound by 2. So, 2 rather than 2 /max. The max is a input signal-dependent aspect which is now sort of taken over by the u transpose u in the step size normalization in the LMS update formula. Similarly, for 10% excess MSE policy, the upper bound for the mu bar would be 0.2 rather than in a previous slide 0.2 divided by the sum of the eigenvalues of the auto-correlation matrix, something which is again input signal dependent. But that input signal dependency is sort of taken over by the u transpose u again in the normalization of the step size. So what you see here is very simple tuning rules for the mu bar and so that further simplifies the application of the of the LMS algorithm and hence the normalized LMS version is the popular LMS algorithm that is oftentimes used in practice.

5.22 Slide 27:

The normalized LMS formula has this regularization parameter, the , and we haven't said too much about this so far. You could say that is indeed a regularization parameter. For instance, at some point in time, uk transpose times uk could be a small number, think of. echo cancellation application, for instance, where at some point the loudspeaker signal is quiet and so zero or close to zero, let's say, and then uk transpose times uk could be a small number or zero. And, and that is basically when that the system doesn't have any input signal and that would the worst time to try and tune the filter coefficients or adapt the filter coefficients. Whereas in the formula you would say in that case uk transpose times

uk is a small number or zero. You would take the largest steps in the adaptation procedure which is definitely not the idea. And hence you have this regularization parameter which certainly at those times reduces the step that you take in the adaptation process. So the has to be there to basically put the lower bound for u transpose u, and it can also have a different interpretation which is provided here in the lower half of the slide. It turns out that the normalized LMS algorithm can also be viewed with the update form. It can also be viewed as providing at each point in time the solution to a specific optimization problem which is provided here. So, the normalized LMS vector at time k is the optimal solution for this optimization problem and it has two terms. On the one hand you're searching for a new filter vector w where you have in the first term of the optimization function the distance from the previous filter vector w_NLMS at timek-1, so in a sense you do not want to run away too far from the previous filter vector. So you take the difference between these two filter vectors and you do the squared two norm and that gets multiplied by an , that's your first term in the optimization function. And then the second term of the optimization function is basically the squared ‘a posteriori error’. So you apply the new filter vector to the input signal u at time k and you compare that to the desired output signal. So here's an optimization function. It has the . And the statement is, if you run the normalized LMS algorithm with mu bar equal to one, then at each point in time the next LMS vector will be the solution to that optimization minimization problem. When goes to zero, then the emphasis is basically on the ‘a posteriori error’. So the optimal solution, the next LMS solution will basically the ‘a posteriori error’ and will do so with a minimum change in the filter vector as it is expressed by the first term in the optimization of function. So the normalized LMS algorithm for mu is equal to mu bar is equal to one, and for sort of converging to zero would set the ‘a posterior error’ to zero in each time step with a minimal change in the filter vector. And by increasing the then you're reducing again the change in the filter vector from one time step to the next time step because you're putting more emphasis in the optimization function on the term that expresses this distance between successive filter vector, so that provides an interpretation of the normalized LMS algorithm and, and even more specifically of the regularization parameter that you see there.

6 Chapter 9

Chapter 9

6.1 Slide 8:

Starting point here is the selection of the cost function. At the beginning of the adaptive or optimal filtering, we have to set a filter structure and then a cost function, a distance measure that we're then going to minimize distance measure for the distance between the filter output signal and the desired filter output signal. And we said we're going to use quadratic cost function. This is pragmatic choice because it leads to simple mathematics. Let us now further look into this selection of the quadratic cost function.

Optimal filter design was based on the MMSE criteria, so we're minimizing the mean squared error. And because you have the mean there, you're basically talking statistical quantities etc. And if you turn that into an adaptive filter, as we have seen it for the LMS algorithm, you have to play tricks in a sense somehow get rid of the expect value operations. You could say, or you could ask what we do an alternative approach where we do not start with statistical values and expected value operations. What if we immediately look at the signal samples as we have observed them in our application in our scenario truly. And that leads to an alternative criterion that we will be using here in in later slides, which is not the MMSE criterion but it's referred to as the least-squares criterion. The basic criterion says at time k, let's say we're not going to use expected values, we're just observing the error samples. When a filter with filter coefficients w is used, we're observing the square of the error samples from time 1 up to a specific time k, let's say so summation from 1 up to k of el

squared where the e error samples are the difference between the desired output samples and actually produced output samples by the filter, when the filter has filter coefficients w and is fed with the input signal u, and this is an alternative criterion immediately using observed samples. When the filter is using filter coefficients as defined by the w vector. And the question is can we use this as a design criterion and what does it exactly lead to?

6.2 Slide 9:

An equivalent formulation of the Least-Square design criteria from the previous slide is, is given in this slide and it's basically sort of the linear algebra version of the same thing. If you look at time k, the filter has processed a number of input samples up to time k, and these define input the samples, define input vectors, the bold u vectors as we have defined them previously. So a bold input vector u1 and then u2 u3 up to uk and let's say and with this you have corresponding desired response sequence or desired output signal sequence with samples d1 d2 up to dk. Now the idea is to have a set of filter coefficients w collected in the bold w vector such that the error sequence of error samples e from e1 up to ek squared is minimized and we're expressing this alternatively in a sort of matrix/vector. Formulation the e1 is the error sample at time 1, d1 the desired output sample at time 1 minus the bold u1 vector multiplied by the filter vector and that corresponds to the to the first row or the first equation in the colored formula. And similarly u2 is equal to d2 minus bold u2 vector times, the bold w vector, etc, on you go on to ek. So what you see appearing here is a vector which is referred to as the bold d vector and then a matrix filled with input samples which is the u matrix which gets multiplied by the filter vector w. And then d vector minus matrix u times the vector w is equal to the sequence of error samples which is referred to as the error vector, the e vector. And so our Least-Squares criterion, which is basically the sum of the squares of the entries in the error vector, the bold e vector is exactly equal to or is defined to be the squared two norm of the e vector is the square two norm of basically d minus u times w. And so when we're minimizing this Least-Squares criteria, we're searching for the w that minimizes d minus u, so d vector minus u matrix times the filter vector the w vector. This is actually fully equivalent to solving an over determined set of equations. Imagine that k is equal to 1000 for instance, and the w has 100 coefficients. Then basically the colored equations is representative of 1000 equations in 100 unknowns. You have more equations than unknowns. You cannot solve this set of equations exactly, but you can search for sort of the best possible solution in a Least-Squares sense. So this is the sort of traditional or standard way of solving over-determined sets of equations by minimizing basically the difference between u times w and the right hand side vector, the d vector. So if you go back to your linear algebra courses and look into Least-Square solutions for over-determined sets of linear equations, this is exactly it. So we're minimizing. d minus u times w, we're searching for the w that minimizes the squared two norm of d minus u times w.

6.3 Slide 10:

Minimizing this Least-Squares criterion is actually very simple and here again you have a closed form solution for the optimal solution, which is obtained by computing the gradient and then setting the gradients of the optimization function to zero. And if you do so, what you see appearing there in the expression for the gradient is, on the one hand, a matrix which is Xuu defined to be U transpose U. And then next to that which is appearing as a vector which is denoted here as Xdu which is U transpose d. So a matrix Xuu and a vector Xdu and then setting the gradient equal to zero really means that in order to compute the optimal solution, the least square solution, will have to solve a set of equations, L+1 equations in L+1 unknowns the entries of the w vector and this set of equations is referred to as the normal equations. There's a closed form solution or closed form formula for the solution, I should say the least square solution is then the inverse of the matrix Xuu times the vector Xdu but again, we're not solving sets of linear equations by doing or computing a matrix inverse. We have cross-elimination and whatever to do this, but anyway, it leads to a simple set of equations referred to

as the normal equations based on this U transpose U matrix, which is defined as the Xuu here, and then the U transpose d vector, which is defined to be the Xdu matrix.

6.4 Slide 11:

What you've seen in the previous slide obviously very much looks like what we have seen in the previous chapter when we did Wiener filter or optimal filter design, where we had a similar matrix Xuu. Or we saw a similar matrix Xuu appearing in a similar vector Xdu appearing in the derivation of the optimal filter vector. Notice that in the previous chapter we have been using a notation with overbar. We had Xuu overbar and Xdu overbar to indicate stochastic quantities. Whereas here in the previous slide we do not have these over bars where we have quantities which are basically computed from observed signal samples of the u signal and the and the d signal immediately. Now the expression for the Xuu in the previous slides as it was given there, it's basically the U matrix transpose times the U matrix you if you rewrite that expression you basically see that that's the first formula on this slide that this is a summation over u times u transpose averaged over a time window that runs from time is equal to 1 up to time is equal to k and that in itself up to a scaling provides you with an estimate for the stochastic quantity Xuu overbar obtained through averaging over time. So sort of assuming stationarity and then also ergodicity which would allow you indeed to estimate stochastic quantities by means of time averaging. So the conclusion here is that the matrix that you see appearing in the previous slide, the Xuu in a sense provides subject to assumptions of stationarity and ergodicity, provides an estimate of the Xuu overbar that is used actually in optimal or Wiener filter theory, and same conclusion for the Xdu vector that you've had in the previous slide. So this basically provides under the same assumption of stationarity and ergodicity provides you with an estimate of the cross-correlation vector Xdu overbar as you have used it or seen it in optimal Wiener filter theory. So, that sort of puts up a link between the Least Square solution as we have computed in the previous slide and an optimal filter or a Wiener filter as we have defined it in the previous chapter. If you're sort of addicted to stochastic signals and systems theory and you're a fan of linear filter theory, you could try and apply Wiener filter theory in practice, let's say for an acoustic echo cancellation example and then you know that the Wiener filter, the optimal filter is the inverse of the input signal auto-correlation matrix times the cross-correlation vector. But in your application, you will not be given these statistical quantities. So you would have to estimate them from your observed signals and a way to do that is actually in the first and second formula in this slide. So you could put up an estimate for the Xuu overbar and the Xdu overbar and plug in those estimates in the optimal Wiener filter formula and so then the sort of somewhat strange result is that leads to the Least-Square solution. The Least-Square solution, in a sense, can be interpreted as an estimate for the Wiener filter solution, which is obtained by plugging in estimates for the stochastic quantities, the Xuu and Xdu, which is justified under an assumption of stationarity and ergodicity. So that's all-nice theory in a sense, but the remarkable thing is that the Least-Square solution indeed corresponds to this, but sort of is a solution in its own right to a different optimization problem or a different design procedure based on the Least-Square criterion. So if you do not care about a stochastic interpretation of all of this and you would say I have an application with signals and I observe signal samples and sort of define a design criterion for my optimal filter or Least-Square filter, which only uses these samples, observed samples from the signals in my application, that leads to the Least-Square solution as we have it in the previous slides, and the interpretation as if it were an estimate for the Wiener filter is nice, but it's in a sense not truly needed. And so in a sense you can safely forget about how Wiener filter theory that we have seen up till now and only stick to the Least-Square solution, it being a solution to the minimization of the Least-Square criterion.

6.5 Slide 12:

Final word here on this, obviously if k goes to infinity, so if you're basically estimating stochastic quantities like Xuu overbar by averaging over an infinitely long time window, then the estimate converges to the true thing. So, the quantity Xuu that you use in the Least-Square solution is actually exactly equal to the Xuu overbar stochastic quantity. So for infinitely long time window the computed Least-Square solution would be exactly equal to the Wiener filter solution, again subject to stationarity and ergodicity assumptions.

6.6 Slide 13:

So, the conclusion is repeated here, the link between Wiener filter optimal filter theory and basically Least-Squares estimation as we have seen in previous slides, says in a practical application statistical information, namely the auto-correlation and cross-correlation quantities will be missing. And so in order to do Wiener filter theory or apply Wiener filter theory you would have to estimate these quantities from the observed data and that requires assumptions basically stationary and ergodicity and then if you compute the Wiener filter solution based on these estimated statistical quantities, then it turns out that the computed, the estimated Wiener filter solution is exactly the Least-Square solution, but then the Least-Square solution in itself optimizes a different criterion, the Least-Square solution, where there's no need whatsoever to rely on statistical assumptions of stationarity and ergodicity. So it's sort of a shortcut to the same solution.

6.7 Slide 15:

Last step now is to derive an adaptive filter through adaptive filter algorithm from this Least-Squares design criterion or Least-Squares estimation from the previous slides. This is somehow similar to the derivation of the LS algorithm from Wiener filter or optimal filter theory and it will lead to an algorithm which is referred to as the RLS algorithm, the recursive Least-Squares algorithm with an update equation which is somehow similar to the updated equation indeed of the LMS algorithm. The underlying problem statement is as follows. In previous slides we have seen the Least-Squares estimation problem where we use observations of, for instance, in the first formula the desired output signals samples d1 d2 up to dk and we've collected these samples in a bold d vector. Now we're adding a time index, a subscript to this bold d vector. So we say bold d vector at time k and similarly we're going to have the U matrix filled with input signal samples for the filter, the bold u vectors. So those define the matrix u, which now also receives or gets the time index, a subscript u at time k, so at time k you can solve this over-determined set of equations or Least-Squares estimation problem and it leads to a solution as we have seen in previous slides, which is basically the Xuu inverse times the Xdu vector. And Xuu is basically U transpose U at time k, which then gets inverted and that is multiplied by U transpose d. And now you could compute such a filter vector at each point in time k, but the basic question is can you organize things recursively, can the solution as you have it here for w at time k and it somehow be computed more efficiently, then also from a previous filter vector which you have computed at the previous time k-1 and that leads to recursive update formula and a truly adaptive algorithm, as you will see in the next slide.

6.8 Slide 16:

Now the formula for the Least-Square solution has two ingredients, the Xuu matrix and the Xdu vector. If you move from time step k-1 to time step k, you can check that you have a simple update equation for the Xuu as well as for the Xdu. Xuu at time k is basically Xuu at time k-1 but there's one additional row, which has been added to the U matrix which is the bold u vector at time k and so if you compute U transpose U at time k, it basically the U transpose U at time k-1 plus the

extra row which is multiplied by its transpose. This is actually referred to as a rank one update because the Xuu at time k is equal to the Xuu at time k-1 plus a vector times its transpose and that vector times its transpose is a matrix of which the rank is one and so it's referred to as a rank one update. And you have a similar update equation for the Xdu vectors. So the next Xdu vector at time k is a previous one at time k-1 plus basically the u vector times d at time k. So here's the update equation for Xdu and Xuu but it's actually the Xuu inverse that we're going to need in the formula or that is used in the formula to compute the Least-Square solution. Now there's an ingenious thing which is called the 'matrix inversion lemma'. If you want to check it out, Google for Wikipedia 'matrix inversion lemma' which basically says that if you update a matrix by means or with a rank one matrix, as we have it here for the Xuu, then when you consider the inverse of that matrix, you would also see a rank one update, so the inverse of Xuu at time k will be equal to the inverse of Xuu at time k-1 and then plus or minus an additional matrix which basically is again a rank one matrix. So what you see in the first formula here, Xuu inverse at time k, Xuu inverse at time k-1 minus a matrix, and in this term, you have a vector. That's the thing between brackets and you can ignore it for the time being. But then you have a vector k times its transpose. And that is again the rank one matrix. So indeed, the inverse also is updated by means of a rank one update. The update vector, or the vector that defines this rank one update the k vector is defined here to be Xuu[k-1] inverse or the previous inverse of the auto-correlation matrix, estimated auto-correlation matrix times the input vector u at time k. So with the previous inverted Xuu multiply with uk, that gives you the k vector. With this k vector, you update the inverse. Notice that usually computing an inverse matrix, inverse from scratch, if the dimension of your matrix is L+1 times L+1, computing such a matrix inverse which require L+1 to the power 3 operations, whereas here you have a recursive formulation, the inverse is computed from the previous inverse. With a rank one update and basically to rank one update, you compute a matrix and then add a matrix to another matrix. Computational complexity of that is basically the dimension of the matrix L1 to the power 2, so that reduces the computational complexity of the matrix inversion significantly. So here we have, the updates of the matrix inverse. If you substitute that in this filter vector formula, you end up and we're not going to do this derivation. So you have to believe us that it leads to this expression. The expression is actually very simple. It says the Least-Square solution, at time k is equal to the previously square solution at time k-1 plus an additional term, a correction term which has two ingredients. It's basically first ingredients, a vector, and this vector is referred to as the 'Kalman gain vector'. We're going to see the name also in later chapters, just a name here. So this is a vector. And that vector and by the way it's equal to Xuu inverse times again the input vector uk that vector gets multiplied by the second ingredient, the factor dk minus uk transpose w_LS at k-1 which is something we have come across already in a previous slide when we looked at the LMS algorithm. This is the 'a priori residual', so basically filtering the input signal with the previous Least-Squares set filter coefficients and comparing the filter output for the desired filter output. So the update equation is getting a very simple one, and the next Least-Square solution is the previous Least-Square solution plus correction, which is in the direction of the Kalman gain vector times or modulated by the 'a prior residual'. Notice that this update equation for the filter vector is an equation similar to the LMS equation but that the update vector is Xuu inverse times the input signal vector bold u where in the LMS algorithm it was only the u vector here, hence the u vector pre-multiplied by Xuu inverse which means that we have to keep track of indeed the Xuu inverse and hence we have to also sort of use the first formula that computes the Xuu inverse from a previous such inverse. And so you could say whereas LMS is a one-line algorithm, this is a two-line algorithm where the second line is the update equation for the filter vector, but it also requires the first line which is the update equation for the matrix inverse, The first line is a matrix equation and the second line is a vector equation, so the first line is basically where all of your computational complexity goes. So the first line is matrix, is equal to matrix plus a matrix basically, and the computation complexity of that is basically order L to the power 2 where the dimension of the matrix is L+1 times L+1, so that's a complexity which is order L squared, in a sense we have a reduction of computational complexity from order L to the 3 to order L 2, if you were to compute the matrix inversion from scratch so you could say this is an efficient algorithm, complexity order L squared, mainly because of the first line of the algorithm compared to LMS which had only the second line vector equation of complexity order L. This is more complex, this is order L squared complexity compared to order L complexity for the LMS algorithm. Notice also that in the

comparison with LMS, the update equation for the filter vector w, a vector at time k is the w vector at time k-1 plus an update term which has a direction for the update which is now the Kalman gain vector and then times the same ‘a prior I residual’. So the only difference in a sense with the LMS algorithm is that the update vector or direction for the update is rather than just that input filter vector to uk, instead input filter vector uk pre-multiplied by this inverted auto-correlation matrix. You can then ask yourself when does that actually corresponds to the LMS update equation. Obviously when Xuu is an identity matrix, then Xuu inverse is an identity matrix and it would basically disappear from the update equation. The result would be basically the LMS update equation so you could conclude that the LMS update equation is basically the RLS update equation when Xuu is an identity matrix. Remember we have considered that case also in the LMS algorithm analysis when Xuu is an identity matrix which happens when the input signal is a white noise signal. Then the conclusion here is that the LMS algorithm is indeed basically an RLS algorithm, and earlier we have concluded that in that case in the LMS algorithm has good convergence properties. Because essentially it corresponds to recursive least squares algorithm. So here we're recursively computing least square solution notice also compared to the LMS algorithm, there's no step size parameter, so step size parameter is sort of in the LMS philosophy you take a step in a direction and you're not exactly sure about the estimated gradient so you reduce the step by the step size parameter mu, there's no step size parameter here in the RLS algorithm. At each point in time your solution is the exact solution to the least squares problem formulated at time k.

6.9 Slide 19:

With the recursive least squares algorithm, we have a mechanism for recursively computing the least squares solution at each point in time k. Now if you look into the definition of the least squares problem that we're solving at time k, it involves observations of the filter input signal and the desired filter output signal from initial time 1 up to time k. Now, especially in a time varying scenario where you're using a filter that has to, in a sense, adapt to time variations in your scenario, it may not be very wise to take very old observations at the initial time 1 and 2, etc, into account. It makes more sense to focus on the more recent samples, and these are Least-Squares filter design only on the more recent samples rather than on samples from a distant past. Now a simple mechanism to introduce that aspect, sort of a simple feature added in the recursive least squares algorithm is referred to here as exponential weighting. And it involves a very simple modification of the least squares criterion. The least squares criterion was previously given as the summation over all time steps up to time k, so L runs from initial time 1 up to time k, and then you have el squared. So the error sample at time l to the power 2. Now what we're adding here is a factor. So each el squared gets a factor which is to the power 2(k-l), where the will be well-chosen number between 0 and 1. The exponent, basically the k-l, sort of tells how old a sample is, so how far it is from the current time k and hence the k-l. So if you choose the to be a number smaller than one, let's say it's going to be 0.9 or 0.99 or 0.999 or whatever. The effect is that in the least squares criterion, the most recent observation, which basically contributes the ek to the power 2, gets multiplied by to zero and that is equal to one, whereas the previous observation e_k-1 to 2 gets multiplied by a to 2 and then a previous previous observation gets multiplied by to 4 and then to 6 and then to 8 etc. And with number smaller than one, these sort of to a certain power gradually get smaller and that means that in the least squares criterion you're putting less emphasis, you're giving less weight to samples, specifically the older samples. The older sample is, the smaller the weight it receives in the least squares criterion. So is a well-chosen weighting factor if you explore out the least squares criterion in sort of the linear algebra matrix version, you see sort of a new definition for the bold d vector at time k, which has successive powers of the , to the 0 for the most recent observation dk, and then to the 1 it will be for d_k-1, and the least squares criterion here because of the squares of the two norm that we're going to consider later on. So basically to the 2 in the last row to the 1 in the previous row, to the 2 in the previous previous row, etc., for the definition of the bold d vector and similarly for the redefined U matrix, u at time k. As to the 0 multiplication in the last row and then to the 1 multiplication in the one but last row, etc. So this is our redefined least squares criterion at time k. Obviously with this redefined d vector and U matrix, you can still do a closed form solution for

the filter vector, which is U transpose U inverse times U transpose d, the same formula. The question is what does the recursive version of this formula looks like and that is given in the next slide.

6.10 Slide 20:

And then the remarkable thing is that this addition of the really doesn't change much in the recursive least squares algorithm. In the second equation of the RLS algorithm, the update equation for the filter vector, basically nothing changes, whereas in the first formula, the update equation for the inverse Xuu matrix, you see a 1/(ˆ2) appearing. It's to the 2 because it's basically correlation U times U, where the U gets multiplied by , so hence to the 2 and it's 1/(to the 2) because you're computing a matrix inverse. So how that makes sense. So basically only in the update equation for the matrix inverse, you see the appearing minor changes in the RLS algorithm and that adds this additional feature where you basically take only the most recent signal observations into account, and you sort of safely forget about all observations that shouldn't play a role in the computation of the least squares filter vector.

6.11 Slide 21:

So, this is our recursive least squares algorithm with the additional features of exponential weighting. Something to remember is that it has a computational complexity which is order L-squared. Where L+1 is the dimension of the vectors or L+1 times L+1 is the dimension of the matrix that you see appearing in the problem formulation. So complexity is L-squared, because we have matrix operations basically, and this can be compared to that complexity of the LMS algorithm or normalized LMS algorithm which is only on the order of L. So complexity for LMS is an order of magnitude, you could say smaller than computational complexity for the RLS algorithm, but then the LMS algorithm can suffer from a very slow convergence. Whereas the RLS algorithm at each point in time gives an exact solution to a specific optimization problem, the least squares optimization problem as you have to find it for time k, so it's more complex than the LMS algorithm. Nevertheless, in the next chapter 9 we're going to look into so-called fast RLS algorithm which give exactly the same solutions as the original RLS algorithm but do so with a computational complexity which is order L, slightly larger computational complexity still compared to the LMS algorithm.

6.12 Slide 22:

Final word on the RLS algorithm. The version of the RLS algorithm as you have seen it in previous slides is the version of that is often given in textbooks. But actually, it turns out that from a numerical analysis and numerical stability point of view, this is not an ideal algorithm especially so when you run the algorithm or you implement it with low precision. So similar to the things we have studied in chapter 6 of the course. So in finite precision implementations, if you will drum this RLS algorithm, it will turn out that it has an unstable quantization error propagation mechanism. That means that suppose you run this RLS algorithm on the one hand with infinite precision, on the other hand with finite precision. Then the idea is that the finite precision implementation provides sort of roughly the same results as the infinite precision algorithm, but this is not the case. After a while, the finite precision algorithm will produce results which are away from the infinite precision algorithm results. And as you go along the results in the finite precision algorithm will be more and more away, further away from the infinite precision results. So you have an unstable quantization error propagation mechanism. We're not going to go into the details, but that calls for an alternative algorithm which is better behaved numerically and that also will be studied in the next chapter we will see alternative algorithms, ‘square root RLS’ algorithms which are perfectly stable even in finite precision implementation, and they have the same complexity and the same convergence properties basically as the initial RLS algorithm. So those are the algorithms basically of choice in practice.

6.13 Slide 24

So, as it has been stated already, the standard recursive least squares algorithm has been shown to have unstable quantization, error propagation and low precision implementation. The meaning of this is that if you would run the standards RLS algorithm, on the one hand in infinite precision, and on the other hand, in finite precision. The two algorithms are fed with the same input signals. And the question could be do they indeed produce the same output signal? And then it turns out that after a number of iterations, basically the finite precision algorithm or implementation will produce results which can be quite far away from the infinite precision results due to round off, for quantization error propagation in the algorithm. we're not going to go the detailed analysis of this round off. We looking to alternative solutions based on orthogonal transform and QR decomposition as you will see it that immediately to avoid such numerical quantization error propagation. So our better outcomes from a numerical point of view, the starting point for this derivation will be the least squares estimation, which then in the second step will be turned into a RLS estimation based, as you will see on orthogonal transformations and QR decomposition.

Starting point will be the least squares estimation problem as we have seen it in the previous chapter. So we're minimizing over a filter coefficient vector. The w minimizing the square two norm of d minus u times w, so we have derived this in the previous chapter. This is our starting point. In the previous chapter, we have also seen that for the optimal w we have a closed form formula. If you remember, it was actually the inverse of (u transpose u), times u transpose d. Now if you would solve this least squares problem, which corresponds to solving an overdetermined set of linear equations in the least squares formulation. If you would enter this problem into software package like MATLAB you could ask what does MATLAB do? How does MATLAB compute the optimal solution? Does it indeed compute u transpose u and then the inverse and then multiply this with u transpose d? The answer is no, this is actually avoided forming the u transpose u and then computing the inverse from a numerical point of view. It turns out not to be a very good idea, so MATLAB will use a different approach. In this approach, which will lead to so called square root algorithms are based on orthogonal matrix factorizations namely the QR factorizations or QR decomposition that you may remember. The statement is if you have a u matrix, you can always decompose that into a product of a Q matrix and an R matrix. And that is referred to as QR decomposition where the Q and the R have special properties. The Q will be an orthogonal matrix. So Q transpose Q is an identity matrix which basically means that columns of the Q matrix form an orthogonal set of basis vectors, and then the R matrix is an upper triangular matrix. There are basically two forms for the QR decomposition, you can define the Q to be a square matrix. So then Q is a k by k matrix and then and then the R part will basically be the rectangular matrix with the same dimensions as the u matrix. Notice that we're assuming that u has more rows than columns. So in a sense it's a thin matrix. Then the R part is also a thin matrix which has the upper triangular part and then everything below the diagonal is zero. So that's the first formulation of the QR decomposition. Q is a square orthogonal matrix and the r part is basically matrix with the same dimension as the u matrix with an upper triangular or L part and then a zero part. The zero part obviously multiplies the last so many columns of the Q matrix, if you leave that out then then basically you have an alternative formulation with only the first L plus one columns of the Q matrix which is denoted here is Q tilde and then multiplied by a square triangular matrix, or the dimension of the triangular matrix is (L+1) times (L+1).

6.14 Slide 25

Everything you need to know about QR decomposition is that every matrix, for example given u matrix here, there is an example with four rows and three columns, can be decomposed into a Q matrix and a triangular R matrix. So here's a simple example. If you enter your matrix into MATLAB and then ask MATLAB to produce a QR factorization, it will return the Q matrix and R matrix as you see it here. As I mentioned in the previous slide, the columns of the Q matrix basically form an orthogonal basis or a set of orthogonal basis vectors. They basically span the columns space of the u matrix. There's

a procedure to derive an orthogonal basis from the column space of a matrix which is referred to as Graham Schmidt procedure. The QR decomposition is representative of that Graham Schmidt process. That is a side remark. Another side remark here is that in the expression for the least square solution, remember it had u transpose u, this u transpose u because of the orthogonality of the Q matrix is equivalent or equal to R transpose R, so u transpose u is basically a short fat matrix times a tall thin matrix. Here you express this as a square triangular matrix transpose times a square triangular matrix R. So if you form u transpose u then R is referred to as the Cholesky factor or square root of the u transpose u. It's the algorithms that we're going to see in later slides, which are indeed referred to as square root algorithms as they work with this R factor, the square root of u transpose u.

6.15 Slide 26

This slide shows how you can exploit the QR factorization to reformulate the least squares estimation problem into an alternative simpler estimation problem. So we're exploiting the QR factorization of the u matrix from the previous slide. We're reconsidering the least squares estimation problem, the minimization over optimization variable, the filter coefficient vector w, and then we're minimizing the square two norm of d minus u times w. Now a two norm is preserved under an orthogonal transformation. The two norm is basically sort of the length of a vector and if you apply an orthogonal transformation, like rotation for instance, to the vector, the length of the vector is unchanged. So this d minus u times w vector can be transformed by an orthogonal transformation and the orthogonal transformation that we use here is the Q transpose. So, the estimation problem is unchanged if we sort of plug in Q transpose and then Q transpose as it applied to the d as well as to the to the u. Now we know from the QR factorization if you compute Q transpose times u matrix, you end up with this rectangular matrix with the triangular part at the token and the zero part at the bottom. So the u matrix is replaced by a matrix with the triangular matrix R and then zero. And that still gets multiplied by the w vector. Now on the other hand, Q transpose d results in a new right hand side vector which is denoted here by means of the z, which is the first part of that vector. And then the second part, the bottom part of the vector is denoted as the asterisk which means we don't care about the bottom part and it will not play a role further on. So we've reformulated our estimation problem into an alternative estimation problem. Now we minimizing over the filter coefficient the w, the difference between [z; the asterisk] minus [r; zero] times the w. Now what would be the optimal w such that the matrix [r; zero] times w optimally approximates the vector [z; asterisk]? You see the w multiplies the zero part to approximate the asterisk, but whichever w you plug in, zero times the w will always be zero. So that really doesn't play a role in the minimization. The only part that plays a role is R times w which should approximate the z. That in fact, knowing that R is a square matrix, is just a set of equations in the coefficients of the of the w, (L+1) equations in (L+1) unknowns. So the minimization problem reduces to solving a set of linear equations in the w coefficients, which is indicated in the last formula R times the w, the optimal set of filter coefficients, the least square solution vector. R times the w should be equal to the z. So the least core solution is basically R inverse times z. And then if you plug in what the r is, it's basically Q tilde the transpose times u, and the z is Q tilde transpose times the d. That from a numerical point of view, appears to be a better version or a better way of computing the least score solution compared to the original version, where you had (u transpose u) inverse times (u transpose d) instead of (Q tilde transpose u) inverse times (Q tilde transpose d). So, this is our preferred way of computing the least squares solution vector. Notice that when we have to solve a square set of linear equations, the left hand side part of the last formula R times w is equal to z, is (L+1) equations in (L+1) unknowns. This is an easy set of equations to solve because the R matrix is indeed a triangular matrix. If you have to solve such a set of linear equations, you're not explicitly going to compute the matrix inverse. You're going to do a simple procedure which is referred to as triangular back substitution. Triangular matrix times a vector of unknowns is a vector of right hand side vector z here. Then because R is a triangular matrix, you can start by computing the last component of the w vector. So that's basically the last diagonal element of the R times the unknown component of the w is the last element in the z

vector. So you can immediately compute the last element in the w vector and then you continue from bottom to the top. That is triangular back substitution, a simple procedure to compute this solution to a set of equations.

6.16 Slide 27

The procedure of the previous slide is also the procedure that is used in commercial software like MATLAB to solve a generally least square problem. So a general least square problem would have a matrix A and the right hand side vector b, in our case, this is the matrix u and the right hand side vector d. If in MATLAB you would enter a command, which is basically b divided by A, and that is used to compute the least squares solution to the least squares problem defined by the A and the b. The algorithm that sits under this command MATLAB indeed uses a QR factorization of a matrix and then a procedure with back substitution as you've seen in the previous slide. So that is everything we have to say about least squares estimation. But now what we actually want to do is recursively squares estimation. So the question is can we turn that whole procedure into a recursive estimation procedure and that will be based on recursive QR factorization which is also referred to as QR updating.

6.17 Slide 28

So in a recursive formulation, we would like to compute that least squares solution at time k from the least square solution at the previous time (k-1). So imagine at time (k-1), we had a u matrix which has a subscript now, u at time (k-1) and the corresponding right hand side vector the boldfaces d at time (k-1). Imagine at time (k-1) we have computed a QR factorization, which basically says you can reduce the u and d by applying the Q transpose at time (k-1) to a triangular matrix and a corresponding right hand side vector in the boldfaced z vector at time (k-1), and then by running a back substitution, which is made explicit here as R inverse times z. But this is basically a back substitution operation for the least squares solution at time (k-1). So if you then go from time (k-1) to time k, the question is do you have to redo the complete QR factorization so that it would give you and a new triangle of factor R and a corresponding right hand side vector z at time k and then you would run a back substitution. The question is do you have to start from scratch where you to redo the whole QR factorization or can you somehow compute the least square solution and so the R and the z from the R and the z of the previous time step. It will turn out that this is indeed very much possible.

6.18 Slide 29

It will turn out that indeed you do not have to recompute a full QR factorization at time k. In fact, at time k, you only have to compute a QR factorization of a much smaller dimension which is indicated here by the last formula. Actually you would construct a matrix which consists of the R matrix from the previous time step R at time (k-1) and you add an extra row which is the boldfaced u vector at time at time k. So this is basically an (L+2) times (L+1) matrix. So it's a square matrix plus one extra row. And of this matrix you compute a QR factorization. So we define the Q matrix here as a square matrix and that leads to a rectangular matrix with a triangular part which will be the new R matrix together with a zero row. So if you go back to the first formula, now you basically construct the same matrix R[k-1] with an additional row, the boldfaces u vector at time k. And if you were to apply the computed Q transpose computed from the QR factorization, that would lead to in the left hand side of the equation, the a new triangular matrix R at time k and then and then a zero row. If you then apply the same orthogonal transformation Q transpose to the right hand side part, which is z at (k-1) with the extra observation, desired output sample at time k which is d[k], apply the same Q transpose. That would lead to an updated new right hand side vector z at time k plus again an asterisk part that for the time being will not play a role and will be basically ignored. So the, the message here is that you construct this compound matrix with the R from time (k-1) and

the right hand side vector z, the boldfaced z from time (k-1). You append an extra row with the boldfaced u vector and the right hand side signal sample at time k. And that is pre multiplied with the Q matrix from the QR factorization at the bottom of the slide. And it leads to a new triangular matrix are at time k and a corresponding right hand side z at time k. It will turn out that these are exactly the R and the z that you would also have obtained if you were to compute the QR factorization at time k from scratch based on the large u matrix at time k and a corresponding right hand side vector d at time k. So that leads to exactly the same solution and that can be proved. We're not going to give any proof here, but that is a given fact.

6.19 Slide 30

Update equation from the previous slide can then be used to piece together the full QR updating based recursively squares algorithm, which is then referred to as a square root recursive least squares algorithm. So we have the update formula from the previous slide. Basically we first piece together this compound matrix with our triangular matrix from the previous time step together with a boldfaced z vector from previous time step when we append in an extra row with the boldfaced u vector at time k and the d sample, desired filter output signal at time k. To this we apply an orthogonal transformation, the Q transpose which is computed from QR factorization. The left hand part of the compound matrix and that leads to an updated compound matrix with a new triangular matrix R at time k, and a corresponding new right hand side vector z at time k. And then the bottom row has been turned into a zero part and an asterisk part, which at this point we do not care about. With a new computed triangular matrix R at time k and a corresponding right hand side vector, the boldfaced z at time k, we can do a back substitution which is provided here as R inverse times z. But in practice, you would actually need to do a triangular back substitution, that would lead to the new least squares vector, the least squares vector at time k. This is sort of a two formula recursive least squares algorithm based on QR updating. The first line is a matrix operation and then the second line is a back substitution to compute least squares vector at time k. Notice that in the update formula we work with a triangular matrix R, a corresponding right hand side z from the previous time step. This is basically the only thing we have to remember from all previous time steps or all previous observations. So nowhere we have the large u matrix with all previous observations or corresponding vector, the boldfaced d vector with all previous observations of the desire to output a signal. All information from previous time steps they sort of compressed or in the triangular matrix R at time (k-1) and right hand side vector z at time (k-1), so you could view the R and z at time (k-1) as all previous information, sort of the memory of the past that you need to compute the least squares solution at time k. So the memory of the past is R[k-1] and z[k-1]. You add new observation, the boldfaced u and the d at time k and then recompute a triangular matrix R at time k and a right hand side vector z at time k which is then the memory from the past that you propagate to the next time step. This is how we should view this algorithm or interpret this algorithm. Final note here is that if you want to include exponential weighting, as we have seen it in the previous chapter, to give a smaller weight to old observations and a larger weight to more recent observations, it turns out that you can easily include in the exponential weighting in that update equation. You have to merely multiply your memory from the past with this landa there before you add the new observation with the boldfaced u and the d[k]. Basically at each point in time you have a memory from the past. You multiply it with lambda and then you append you information from time step k which basically again leads to an interpretation where you could say something that came in an observation in 100 time steps ago in the meantime has been multiplied by land to the 100, effectively has been given a smaller weight. So here exponential weighting is again easily included in the update formula.

6.20 Slide 31

In the basic update equation, we have this matrix Q at time k transpose and in previous slide we said, and this Q matrix is basically computed by computing a QR factorization or doing a QR factorization of a compound matrix. If you go back a few slides, you would see the compound matrix with R at time (k-1) and the additional row boldfaced u at time k, and so you compute a QR factorization of that and that leads to the Q matrix that you see in this basic update equation. Now it turns out that because of the special structure of the R at time (k-1), it being an upper triangular matrix and hence with a zero structure in a sense that the Q matrix can also be sort of unraveled or decomposed into a sequence of very elementary orthogonal transformations or orthogonal matrices which are referred to as given rotations. And that will be developed and explained further in in the next few slides. For this we have to define a given rotation, which is one of these elementary orthogonal rotations or orthogonal matrices. A given rotation matrix is defined here as G. This matrix is defined to be an identity matrix except for four entries. And these entries are at the intersection of the i-th and the j-th column and row. So the given matrix basically is specified by an I, a j and then a rotation angle theta, of which the cosine and the sine will appear at the intersection of the i-th and the j-th columns and rows. So at the intersection of the i-th and the j-th columns and rows, you would see cosine of theta, sine of theta, minus sine of theta and cosine of data. The meaning of that matrix is that if you apply it to a vector, you will see the next slide, you basically rotate the vector in the i-th and the j-th dimension over an angle theta.

6.21 Slide 32

This is given more explicitly in this slide. So if you apply the given matrix or orthogonal rotation to a general vector X, and that results in vector X tilde, which is defined here. Then you would observe that all components of the X tilde, the vector is basically unchanged, or the corresponding components in the X vector except for the i-th and the j-th component where X tilde i and X tilde j are a linear combination of the Xi and the Xj components of the original X vector, where the weights in the linear combinations are the cosine and the sine of the rotation angle theta. The meaning of these two formulas is basically organizing an orthogonal rotation over an angle theta in the i and the j dimension. Now by picking an appropriate rotation angle theta, such rotation can also be used to set one of the resulting components in the X vector to zero. For instance, the j-th component Xj can be set to zero if the rotation angle is selected. And then you have the last formula here. The tangent of the rotation angle should be basically the ratio of the j components and the i components in the X vector. So by picking appropriate rotation angles you can basically rotate a vector to the X axis, or the Y axis if you would envisage a two dimensional representation and so set one of its components equal to zero.

6.22 Slide 33

With the definition of the given transformations in the previous slide, we're now going to construct the Q matrix of the basic update equation as a sequence of such a given transformations. Notice that a given transformation is represented by an orthogonal matrix. If you apply a sequence of given transform that corresponds to multiplying a sequence of orthogonal matrices together. And a product of orthogonal matrix is still an orthogonal matrix, and that eventually will be our Q matrix as you see it here in the update equation. So Q is going to be constructed as a sequence of given transformations. Notice that the Q multiplies the compounds matrix with the R[k-1], z[k-1], u[k] and d[k], multiplying that matrix to the left. Multiplication to the left corresponds to applying row transformations to this compound matrix. So the Q corresponds to applying row transformations onto the compound matrix. Now, the Q matrix should be such that if you apply it to the compound matrix, you get a new compound matrix. The left hand side of the first equation with zeros in the bottom row. And the way we're going to construct the queue is such that the zeros will be introduced one by one, from the left to the

right. And this is done by a sequence of given transformations that apply row transformations to the compound matrix where you combine the first row of the compound matrix with the last row and then the second row with the last row and then the third row with the last row etcetera. In order to introduce the first zero in the last row and then the second zero and then the third zero etcetera on you go. This is illustrated here at the bottom of the slide with a small three by three example where the meaning is we have a three by three triangular matrix R[k-1], if you see an X then generally that is among zero element, whereas if you don't see an X, that is a zero entry. So indeed you have a three by three triangular matrix and then a three by one right hand side vector, the boldfaced z. And in the bottom row you have the boldfaced u and the element to the bottom right, which is the sample d at time k. Now first rotation given transformation will be applied which responds to a row transformation where they combine the first row with the last, in this case the fourth row, of the compound matrix. So the first given transformation rotates the first row together with the last row, which means that the (1,1) element of the matrix is rotated together with the (4,1) element, and then the (1,2) element is rotated with the (4,2) element and then the (1,3) element is rotated with the (4,3) element and finally the (1,4) element is rotated together with the (4,4) element. Now the rotation angle of this transformation will be such that and a zero is introduced in the (4,1) entry of the matrix. So if you go from the first compound matrix one step to the right to the second compound matrix, etc., you don't see any X, so zero entry in the (4,1) position of that compound matrix. In the next step you're going to apply the second given transformation, which where the i and the j dimension are basically 2 and 4, so you're combining the second row of the compound matrix with the fourth row, the last row. And again apply a given rotation. So the given rotation will take the (2,2) element of the compound matrix and rotate it together with the (4,2) element and then at the same time take the (2,3) element and rotate with the (4,3) element and finally take the (2,4) element and rotate with the (4,4) element. Now in this case, the rotation angle theta will be such that again a zero is introduced in the (4,2) position of the compound matrix. And then finally in the last step here, we were going to apply a third given rotation which corresponds to a row transformation, combining the third row with the last row, with the fourth row. So you're rotating together the (3,3) element with (4,3) element as well as the (3,4) element with the (4,4) element. Now the rotation angle will be such that zero is introduced in the (4,3) position of the compound matrix. So you see the successive given transformations, each of them being an orthogonal transformation and introduce zeros in the bottom row of the compound matrix one by one from left to right. You also observed that because of the special structure of the compound matrix with the triangular part that once a zero is introduced, it will never be filled by later transformation. So if you introduce a zero somewhere it this zero will continue to be a zero in later steps. So we have a procedure based on give transformations to introduce zeros in the compiled matrix one by one from left to right by combining the first row of the matrix with the last row, which is sort of the row representing the new observations at time k, and then combine the second row of the matrix with new observations, and then the third row of the compound matrix with the new observations, etc.

6.23 Slide34

In the next slide, we'll see a graphical representation of that QR updating process based on a sequence of given transformation. Such a graphical representation is referred to as a signal flow graph. You can view it as actually a realizations that we have seen in chapter four, where you see a filter structure based on delay elements and multipliers and adders. This is going to be similar. The graphical representation here, the signal flow graph in the next slide will be used to derive in a graphical fashion of fast recursive least squares algorithm from this initial QR updating based recursive least squares algorithm.

6.24 Slide 35

This is the graphical representation of the basic update equation with the compound matrix. What we have here is now a four by four examples of the matrix R, and this case is a four by four matrix and the right hand side vector the boldfaced z vector has four components. What you see in the graphical representation is obviously a triangular structure and then an additional column to the right, which corresponds to the z vector. First, the black squares represent basically memory cells as it is indicated to the bottom right of the slide. A black square is a delay element or a memory cell completed with multiplication and the land of the exponential weight factor, in case you want to include exponential weights. So think of the black square as a delay element as you have it in a general filter realization or a memory cell. And so you see a triangular structure of memory cells storing the triangular matrix at a certain point in time, together with an additional column of memory cells to the right, storing the boldfaced z vector at a certain point in time,. What you also see in the single photograph is a hexagon, which in a sense is shorthand for a collection of multiplications and additions that you do to execute an orthogonal rotation. So we have orthogonal rotation and we will define them based on the given transformation definition from a few slides back. Rotation has a rotation angle theta and it takes in two components as you see it here to the right at the top of the slide. Two components, in general an a and a b element, and you rotate them into an a prime, where the b prime or the a prime is basically a times cosine of theta plus b times sine of theta and a similar definition for the b prime. So hexagon is a shorthand for a number of multiplications and additions as you have to do in the orthogonal rotation. You take in a rotation angle of theta and you apply the rotation to an input two vectors a and b, resulting in an a prime b prime, and then the rotation angle theta as you see in the graph is typically propagated further on to the next rotation cell. So back to the graph, what you see is the structure, right hand side column, R and the z part and now the new observations, the boldfaced u vector together with the right hand side sample d at time k run into the signal flow graph at the top of the signal flow graph. At the top you see an input sequence of samples u[k], u[k-1], u[k-2], and u[k-3]. That is basically the output from a delay line as you have it in an FIR filter. And these four samples define the boldfaced u vector at time k, so this is the input from the top. Then in the right hand column you have an input d at the time k. Now k in square brackets is exactly the same thing with k is as a subscript. So this vector and the corresponding right hand side d[k] observation are fed in from the top and into the signal flow graph. This will define a first rotation given rotation. Look at the top left of the graph where you feed in u[k] together with R11, the current R11 element into rotation cell that also defines the rotation angle. This is your first given rotation to zero the or produce the first zero in the compound matrix and the last row of the compound matrix which indeed leaves the rotation cell as a zero. So you see a rotation cell and input is u[k] and the other input is R11 from the previous time step (k-1) and you rotate these two elements together to produce a zero for the u[k] input and produce an updated R11 at time k for the (1,1) elements of the new triangular matrix. The rotation angle, the theta, that you need in order to produce that first zero is propagated to the rest of the first row. So it's going to go into a rotation cell that takes R12 together with the u[k-1] from the top and rotates them together into an updated R12 at time k. Then an updated element in the last row of the compound matrix which is then further propagated to the second row etcetera. So this first given rotation angle is propagated throughout the first row from left to right. And the rotation is applied to all elements of the first row of the R matrix together with the inputs from the top, the u[k-1], u[k-2], u[k-3] and then also in the right hand side vector to the z11 together with input d[k] from the top. The result of this which in the formula in the top right corner of the slide is the resulted last row of the compound matrix after one given rotation, is then propagated to the to the next row in the triangular matrix and corresponding right hand side vector. So in the second row you will see the second input to the rotation cell that also has the R22 and there you're going to compute the second given transformation with an angle such that the second input is set to zero. And that rotation angle of that second given rotation is then again propagated through the second row from left to right to apply this rotation through the R23 elements together with something that comes in from the top. And the result is propagated to the bottom, etc. Then in the third row you're going to see the third given rotation, and in the fourth row you're going to see, in this case, last fourth given rotation. So from a distance you see a triangular structure that, at the beginning of the

update for time k, stores the R matrix at (k-1), you have the additional column to the right that before the update time k stores z at time (k-1) and then you have the observations, the inputs at time k, which is the boldfaced u vector and the corresponding right hand side where desired output signal at time k, which are fed in from the top and this input ripples through the structure from top to bottom and in the meantime you apply rotation angles to gradually set the elements in the boldfaced u vector to zero from left to right, and in the meantime updates the R matrix from R[k-1] to R[k] and same thing in the right hand side column. Apply the same transformations to update the z at time (k-1) vector to the new z vector at time k. So you see a wave of computations that starts in the first row of the compound matrix, and in the second, third and finally fourth row.

6.25 Slide 37

Remember that the R matrix and the corresponding right hand side vector z are said to be the memory of the algorithm, so at each point in time you have an R and a z. And if you are to compute the filter coefficients, the least squares filter, then you would have to do a back substitution basically corresponding to R inverse times z. So at each point in time you can take out the r and the z and then compute R inverse time z to get the filter coefficients. And so we have the basic update equation and we have the graphical interpretation of that in the previous slide. In the basic update equation there was this asterisk or star which so far we have largely ignored. And also in the signal flow graph in the previous slide you would see that this asterisk or star leaves the signal flow graph at the bottom right of the signal flow graph in the additional column with the z, the end result. All the operations that start at the top and end at the bottom of the signal graph result in this star or asterisk quantity. And so far we have ignored this, we have not considered this or defined it or try to find out what it actually represents. It turns out that this star asterisk indeed has a true meaning in terms of least squares residual and can be used in some applications where the explicit knowledge of the least squares filter vector is actually not needed. And we're going to see an example of that in the later slide.

6.26 Slide 38

If we go back to this basic update equation with the compelled matrix and we give the asterisk or the star a name. Now we name it epsilon, then you have the formula at the top of the slide here. And then it turns out that this epsilon is sort of a very interesting quantity. The last formula here actually gives an expression for the a priori residual which we have seen. We have come across in for instance, the LMS update equation or the RLS update equation. So the a priori residual is basically the residual, the difference between the desired outputs at time k and the filter output when you use the filter coefficients from the previous time steps. So in this case the least squares vector at time (k-1), if you apply that filter to the filter input signal, you compute boldfaced u times the filter vector at time (k-1) and subtracts the results from the desired output signal d[k], that is the defined a priori residual. This is a bit of a strange formula but it's actually not too difficult to prove this even though proof is going to be left out here that a prior residual is equal to the epsilon divided by, and this is remarkable part, the product of the cosines of all the rotation angles that you have used in the last update process with all the Givens transformations. So with all the Givens transformations you sort of take the theta rotation angles and then the cosines them off, and you multiply them together. That leads to the product of the cosines. You divide the epsilon by the product of the cosines and that is the a priori residual. Similarly to the a priori residual, you can also define an a posteriori residual where you compute residual based on the updated set of filter coefficients. So, this is a formula in the middle where you compare d at time k to the output of your filter with the new set of filter coefficients w at time k. That is referred to as the a posteriori residual. And then it turns out again sort of remarkable but easy to prove that this is equal to the epsilon now multiplied by the product of the same cosines of the rotation angles of the last update process with the Givens rotation. So if you compare the last two formulas, you can also define epsilon or view that as the geometric mean

of the a posteriori and the a priori residual. This is remarkable. And it can actually be used in specific applications, as you will see in the next slide.

6.27 Slide 39

Then the computation of the a posteriori or a priori residual can actually easily be added to the signal flow graph. Remember that in the signal flow graph you would see the epsilon appearing at the bottom of the right hand side column with the z vector. So the output there is the epsilon and then while computing the Givens rotations from top to bottom, you could also from the cells, compute the rotation angles, extract the cosines of the rotation angles, multiply them together and then in the end multiply the epsilon signal with the product of the cosines. That would lead to the a posteriori residual. And this operation or this process is referred to as residual extraction. And the ingenious thing is that you do so without computing the filter coefficient vector. So in the formula for the a posteriori residual as you also have it repeated here on this slide, the a posteriori residual is defined based on the least squares filter coefficient vector at time k. So if you would freeze the adaptation, extract the R and the right hand side vector z, do the back substitution, compute the filter vector at time k, then you could compute the a posteriori residual. But the message here you don't even have to do this. You do not have to compute the filter coefficient w explicitly. You can compute it from a by product, the epsilon output from the signal flow graph, from the update process with the sequence of the Givens rotations, and then the cosine of the givens rotations etcetera. And similarly, the a priori residual, again without explicitly computing the filter coefficient factor, can be obtained. And that process which is very useful in some practical applications, is referred to as residual extraction.

A fully equivalent signal flow graph, fully equivalent to the one in the previous slide is provided in this slide. For instance, you would like to see only rotation cells in your flow graph rather than the multiplications for the cosines etcetera. If you just add an additional column of rotation cells as you have it in this slide, which is fed with basically an identity vector, one is fed in from the top, and zeros and the rows further down, you can easily check that the hexagon in the top right corner of the signal flow graph takes in a one and a zero. You apply the rotation angle comes in from the left. Then the result, which is propagated to the next row, will actually be the cosine of the rotation angle. And then in the second added rotation cell, you would again multiply with the cosine of the second rotation angle, etcetera. So in the end, in similar fashion, you obtain the product of the cosine, cosine of all the rotation angles, exactly the same result as you had in the previous slide. But now the nice thing in a sense is that you see only rotation cells.

6.28 Slide 40

An example, or an application of residual extraction is given here in this slide. It refers to acoustic echo cancellation. Remember for an acoustic echo cancellation, we need an adaptive filter to model an acoustic coupling, a transfer function from a loudspeaker to a microphone. And so the filter takes an input loudspeaker signal, you see the loudspeaker signal that is fed into a delay line. Delay line refers to the FIR filter. In this case we give a very small example with only three delay elements. So a third order FIR filter. In practice, if you do acoustic echo cancellation, you would have hundreds or thousands of delay elements. So it would be a much bigger figure. But the principle is the same thing. So the input to the triangular part of the adaptive filter is the boldfaced u vector that appears from a delay line as always, but the delay line is fed with a loudspeaker signal and the desired output signal. If you go back to the explanation of acoustic echo cancellation in chapter 7 then you remember that the desired output signal is the microphone signal here. So that goes into the additional column that will be our d signal. If you do residual extraction then you're extracting residuals. The meaning of the residuals is basically d minus u times w, d is the microphone signal. What is subtract from the microphone signal is the filtered version of the input signal, the filtered loudspeaker signal. And the filtering is such that it models the acoustic path from the loudspeaker to microphone. So if you have a good model for that acoustic path, if you filter is a good model

for that acoustic transfer function, then you filter the loudspeaker signal with the model of your acoustic transfer function. That models the echo contribution in the microphone signal and is indeed then subtracted from the microphone signal. So that echo cancelled microphone signal does indeed in this case corresponds to the least squares residual, being either a priori or a posteriori residual. So in this case, extracting the residuals is sufficient for the application. About the actual model for the acoustic coupling between the loudspeaker and the microphone, the actual filter coefficients, you don't care. The only thing that counts is the residual signal, which is the overall output of the echo cancellation operation.

7 Chapter 10

Chapter 10

7.1 Slide 9

The standards recursively squares algorithm from the previous chapter and also the square root QR based recursive least squares algorithm in the first half of this chapter have a complexity which is order L squared per time update. So per time update, for per sample you would have to do a number of multiplications and additions which is on the order of L to the power two. Now for many applications this is a too large computational complexity. For instance, in the previous slide, the acoustic echo cancellation application, the order of the adaptive filter would typically be hundreds or even a thousand or so. And then L to two is a too large number in terms of the multiplications you would have to do in each sample period. So then the question is can you reduce complexity to linear complexity, linear in the filter order, order L similar to what we have seen for the initial LMS algorithm. LMS had complexity which is linear in L and you would like to have the same thing for RLS algorithms. Now it turns out there are indeed a number of so called fast RLS algorithms. There's a whole collection of them. We're going to consider only one namely the QRD least squares lattice algorithm which will be derived from the QR RLS algorithm from the previous part of the chapter. And this algorithm, as it is based on a procedure with orthogonal transformations, is actually also from a numerical point of view a very good algorithm. Some of the other fast algorithms also do not have good numerical properties, so could have numerical quantization, are in a sense algorithms which sometimes are avoided in practice. Not so for the QRD least squares algorithm, it's perfectly numerically stable and it's a fast algorithm.

7.2 Slide 10

If you were to visit the library and check DSP textbooks or DSP journals on vast least squares algorithms, you would see that there's a vast literature available and that most of this is highly mathematical. In the next slide I show an example of an algorithm and you will see lots of subscripts and superscripts and indices. It gives you a headache. And so it's highly mathematical involved and in a sense we would like to avoid sort of that level of mathematics. So what we're going to see in the next few slides is basically a graphical representation or graphical derivation of a fast algorithm, this QRD least squares algorithm, by manipulating the graphical representation signal flow graph of the initial QRD updating recursive least square algorithm. And by using only the signal flow graph graphical representation, rather than doing a mathematical derivation, it is believed that this actually provides more insight into the derivation as well as into the algorithmic structure resulting from the derivation.

7.3 Slide 11

An example of one fast recursive least squares algorithm is provided in this slide. And the idea is not so much to look into the details of the formulas that you have in this algorithm. The idea is only to sort of give you an idea of the type of headache you may get when you start looking into the mathematical derivations and results of the literature on fast RLS algorithms. So I'm not going to do any mathematics here. But in fact, the algorithm that we're going to drive in the next few slides in a graphical way is indeed the algorithm which is provided here. So this algorithm, you can forget about the mathematics, but a graphical representation will be the result of the derivation in the next few slides and will be given in slide 37 actually.

7.4 Slide 13-15

Before we can start with the derivation of the QRD least squares lattice algorithm, we first have to indicate a property of the least square residuals. That will be given in the next slide, slide 32. That property in itself will actually lead to a system for giving names to intermediate signals that flow in the signal flow graph as we have seen it earlier. And so we will provide a system of epsilon notation to specify what an intermediate signal flowing in the signal flow graph truly means, as an epsilon signal of some embedded smaller least squares problem, as you will see it in the later slide, slide 33. So details are provided in the next two slides.

7.5 Slide 14

In the previous slide we have seen how least squares residuals can be extracted from the signal flow graph representing the update process with the compound matrix. So this is repeated here. Here's a five by five triangular matrix example. On the right hand side product of the cosine accumulation is represented to the left of the signal flow graph that produces the epsilon and the product of the cosine. You multiply the two together, you get the a posteriori residual. If you divide one by the other you get the a priori residual. So this is how we compute least squares residual. Now the property which is indicated in this slide is that if you were to permute the input signals to that signal flow graph, then the question could be what happens to the least squares residuals. And the conclusion will be that the least squares residuals indeed do not change. So if you go to the top of the signal flow graph, you have to filter input signal the u[k] that is fed into a delay line producing u[k-1], u[k-2], u[k-3], u[k-4] and then usually these signals which are produced by the delay line are straightly fed into the triangular matrix. But now what we have done is we've inserted a random but fixed of permutation that takes the u[k], u[k-1], u[k-2], etcetera and permutes them into another vector which is then fed into the triangular matrix. So the u[k] will be fed into the second column, the u[k-1] is going to fed into the fourth column etc. So you apply a random but fixed permutation to the input vector. Then the question is what happens to the R matrix, the right hand side vector z, and then what happens to the least squares filter vector, and finally what happens to the least squares residuals that are produced by this least squares solution. Now you can easily prove, and the proof is also included in in this slide, that if you apply a permutation to the input signals, which is a column permutation to the u matrix that results into a corresponding permutation of the filter coefficient vector, the w vector. With a permuted filter vector, if you then compute residuals, you have the permutation in the input signals and the corresponding sort of undoing permutation in the filter coefficient vector that leads to exactly the same least squares residuals. So the conclusion is that you permute the input signals, it's not going to have any effect in the least squares residuals and the proof is included in this slide. The remarkable thing here is that the conclusion actually applies to the a posteriori as well as to the a priori residual. Then the ingredients to compute these two residuals are the epsilon on the one hand as you see it also indicated in the signal flow graph and then the product of the cosine. Now if both the product of these two ingredients and the ratio of these two ingredients remain

unchanged, then the ingredients themselves also remain unchanged. So conclusion will be the epsilon is unchanged and then also the product of the cosine will be unchanged. So you permute the input signals that would will lead to different triangular matrix, different updating process, different given rotations etcetera and hence different cosines of the rotation angles. But in the end if you multiply all the products of the cosine of the rotation angles of one update process at time k, for instance, together, that leads to a product of the cosines which is unchanged under the permutation which is remarkable. So long story short that least squares residuals are not changed under the permutation. So the order of the input signals from the delay line doesn't matter, doesn't have an impact on the residuals. The residuals are not changed, the epsilon is not changed, and the product of the cosines is not changed in one update process.

7.6 Slide 16

Property from the previous slide allows us to introduce a naming system or a system for identifying intermediate signals in the signal flow graph. So in the signal flow graph, we have signals running from one row to the next row. And we're going to have a system for identifying these signals. As an example we consider signals running from or between the second row and the third row in this signal flow graph. And first look into the signal that leaves the rotation cell, that also has R24, so the signal leaving there can be viewed as the epsilon signal of a smaller embedded least squares problem with a triangular matrix and a corresponding right hand side vector which are colored here in this slide. So you see a two by two triangular matrix which is colored with the input signals to that triangular matrix or u[k] and u[k-1], and then you see a two-elements column vector, which is fed with an input signal u[k-3] and of which the output is viewed as an epsilon signal and then this epsilon signal will be given further subscripts and superscripts to identify it. But the important part is that that signal leaving the rotation cell, the cell that has R24, is viewed as the epsilon signal of a smaller signal flow graph, in a sense with a two by two triangular matrix and then a two component right hand side vector, the vector which is also colored in the signal flow graph. Now we're going to add a subscript and a superscript to the epsilon notation and it will be like this. We have an epsilon signal and it corresponds to a right hand side or an embedded least squares problem with a right hand side vector, which in this case is fed with u[k-3], and it's the time index (k-3) which is used as a superscript in the epsilon notation. So you see the epsilon notation which sort of the signal is extracted and given a name to the left of the graph. The name of the epsilon sign is epsilon, and the superscript is (k-3), because this is the time index for the signal that feeds the right hand side vector. The triangular part of the embedded least squares problem is fed with two signals u[k] and u[k-1], so timing index is running from (k-1) to k. And this defines the subscript in the epsilon notation. So the epsilon notation has superscript (k-3) and subscript (k-1) up to k, this is like MATLAB notation in index running from (k-1) up to k. Notice that the input signals to this two by two triangular matrix are k first and then (k-1) second, but the order referring to the previous slide doesn't matter. So we only have to indicate the collection or sort of the set of indices. And these time indices indeed run from (k-1) up to k and hence in the naming system for the epsilon signals, (k-1) up to k is a subscript. A second and a third example are added here. If you look into the signal leaving the hexagon, the rotation cell, that has R23. Then you extract this signal and you make it more explicit to the left of the signal flow graph. You view this now as the epsilon signal from an embedded least squares problem where the right hand side vector is fed with u[k-2], so the superscript will be (k-2) and the triangular matrix of this embedded least squares problem is still the same two by two matrix fed with u[k] and u[k1], so the subscript of this epsilon signal is again (k-1) up to k. Last example in this slide, if you look to the signal leaving the rotation cell, that has R34, then this epsilon signal, again extracted and made more explicit to the left of the signal flow graph, is the epsilon signal of an embedded three by three least squares problem. The right hand side vector, the three component vector is fed with input signal u[k-3], so the superscript is again (k-3) and then the triangular matrix is a three by three triangular matrix which is fed with u[k], u[k-1], u[k-2]. So timing indices from (k-2) up to k and hence the subscript of the epsilon is indeed (k-2) up to k.

7.7 Slide 19

Starting point now for the derivation is going to be the original signal flow graph with a few additional operations added. So in this graph, first identify the triangular matrix. This is a five by five triangular matrix fed with input signals u[k], u[k-1], u[k-2], u[k-3], u[k-4] from a delay line. So, we have the triangular matrix, the R matrix and then the corresponding right hand side column which is fed with the desired response signal d[k] at the top. And so that has the z column to the far right of the signal flow graph. What you should also see is at the bottom, you have the product of the cosine of the rotation angles. And this is produced in an additional column of rotation cells which is fed with an identity matrix of one and then all zeros, as we have seen it in an earlier slide. There are two things that have been added here to the signal flow graph, first adding complexity to the signal flow graph and then later be able to reduce complexity in the signal flow graph. The first thing that has been added is the additional column which also has the indication with B, which is an additional column of rotation cells and is fed with, in this case u[k+1]. So at the front of the delay line, we add an additional delay element so that the input to that delay would be u[k+1] and then the output of the first layer would be the u[k] etcetera. And if you shift time indices etcetera, then u[k+1] sort of looking into the future, one step ahead wouldn't be a major issue. So basically we add a delay line so that u[k+1] becomes available and that feeds an additional column. The additional column of rotations cells where the rotations are defined are the same given rotations as defined in the diagonal elements of the triangular matrix. So in the triangular matrix, on the diagonal, you compute Givens rotations and propagate the rotation angle both to the right as well as to the left. And to the left, you have rotations in the newly added column and then also rotations in the product of the cosine column. So that is the first thing we have added here, and you do not have to ask questions at this point. It's just additional operations that we're adding in order, as I mentioned, later to be able to reduce complexity. The second thing that we have added are the additional two rotations to the left of the signal flow graph. Basically we extract two signals from the signal flow graph, the signal leaving the rotation cell that has the R23 and that is a signal which can be identified to be epsilon with superscript (k-2) and subscript (k-1) up two k, as we've seen that in the previous slide. That is a first extracted signal and then we're going to in the same row, extract a similar signal from the newly added column, the column with the indicator B. That signal can be identified to be an epsilon signal with superscript (k+1) because that B column is fed with input signal u[k+1], and then subscript for the epsilon signal (k-1) up to k because basically the rotations applied in that newly added column are computed in the same two by two triangular matrix which is fed with u[k] and u[k-1]. So we have these two signals extracted. In the original signal flow graph you would see that the signal leaving the rotation cell R23 would then go into the rotation cell with R33, where you compute a given rotation angle which is then applied to the left and to the right in the other rotation cells. Now in the added rotation cells to the left of the signal flow graph, we're going to in a sense, reverse roles and take off the two signals that are extracted. We're going to take the other signal to compute rotation, givens rotation, which is then applied to the other signal. So we take the signal, the epsilon superscript (k+1) subscript (k-1) up to k signal and compute a rotation that is in the cell, which is also indicated with indicator D, and that rotation is also applied in a second rotation cell which is fed with the other extracted signal, epsilon superscript (k-2) subscript (k-1) up to k, and that's the rotation cell with E indicator. Now the question would now be what is the resulting epsilon signal from this additional rotation cell with the E indicator. That epsilon signal can actually be identified to be epsilon with superscript (k-2). To understand, you go upwards from this cell with the E indicator and then back in the original signal flow graph upwards in the column with the C indicator, and that column is indeed fed with u[k-2]. So the superscript of the extracted epsilon signal is (k-2). To identify the subscript of this epsilon signal, you would have to check where the rotation angles are computed. And for this you can actually construct a three by three triangular matrix as it is indicated at the top left of the slide with the A, B and D parts as you have them in the full signal flow graph. This triangular matrix with the A, B and D part is fed with input signals u[k+1], u[k] and u[k-1]. So timing indices are from (k-1) up to (k+1). So the epsilon signal appearing from the added rotation cell with indicator E is epsilon with superscript (k-2) and subscript (k-1) up to (k+1). Now the interesting part is, if you now look into the signal which is produced in the original signal flow graph by the cell that has the triangular matrix elements R34, that

signal, leaving that rotation cell is an epsilon signal. As we have actually done in the previous slide, it is an epsilon signal with superscript (k-3) and subscript (k-2) up to k. If you compare the constructed epsilon signal and now I'm referring to the equation as you see it in the color part of the slide, on the one hand you have the epsilon signal which we have constructed with the additional operations, epsilon superscript (k-2) subscript (k-1) up to (k+1), and on the other side of the equation we have the extracted signal extracted from the rotation cell with R34 which is epsilon superscript (k-3) subscript (k-2) up to k, which you see is these two epsilon signals have basically the same subscripts and superscripts up to a minus one, which means that if you take the epsilon signal on the left and you delay it over one sampling period, that would basically change the subscripts and the superscripts. You would, subtract one everywhere. That would lead to the epsilon signal to the right of the equation. So with the additional operations, we've produced an epsilon signal that if we delayed over one sampling period, would produce an epsilon signal which is also available somewhere else in the original signal flow graph. And that will be an interesting observation which will be exploited in the next signal flow graph.

7.8 Slide 20

From the previous slide, we learned that there's actually two ways of producing one and the same signal. We have added additional computations in the signal flow graph and then generated a signal which was also available in the original signal flow graph. Now, if one signal can be produced in two different ways, then of course there is redundance and we can remove one of the computations. That is what is basically done in this graph. So we have the two signals which are equal to each other in the previous slide. And what we now have done is that we have removed the original way of producing that signal and that basically corresponds to a sequence of rotation cells in the fourth column of the original triangular matrix, which now can be removed. So if you compare this signal flow graph with the signal flow graph in the previous slide, you see that the fourth column of the triangular matrix has been removed, except for its diagonal elements. That diagonal cell which has the R44 element and the input to that diagonal cell is now produced by this alternative way of computing it with the added computations. This is basically the main trick in the derivation of a fast algorithm. We have found a way of removing some of the complexity from the original triangular matrix. And if we repeat this matrix in each and every row of the signal flow graph, then we will be able to remove all columns of the triangular matrix except for their diagonal elements, so only keep the diagonal part of the triangular matrix and get rid of the complete off diagonal part of the matrix. So if we apply this matrix to each and every row in the original signal flow graph, we end up with a signal flow graph which is indicated in the next slide.

7.9 Slide 21

So the final step in the derivation already is to repeat the main trick for each and every row and the original single flow graph. So in each row we're now adding two rotation cells to the left of the graph, which will allow us to remove a complete column from the triangular matrix in the original signal flow graph, except for the diagonal cell in that column of the original triangular matrix. That leads eventually to the structure as you see it here in each row. You have a very similar structure and this will be referred to as a layer. In each layer you see six rotation cells. If you start from the original triangular matrix, the only thing that remains is diagonal cells. So one rotation cell in a layer is the diagonal cell from the original triangular matrix. Then to the far right you have a cell corresponding to desired input signal, then going from right to left. Now we have one cell for the additional column. The newly added column with input signal u at time (k+1), then you have two cells for the accumulation of the cosine of the rotation angles, and then finally to the left, you have the two newly added rotation cells to play our main trick.

7.10 Slide 22

In this final single flow graph, you see a number of layers stacked on top of each other. The number of layers is (L+1) for an L-th order filter. In each layer you would see as we counted in the previous slide, six rotation cells. If each rotation takes four multiplications, then the number of rotations is basically 24 per layer with then (L+1) layers. So the complexity counted in number of multiplications would be 24 times (L+1). This is still larger complexity than a complexity for LMS which was also linear in L, but in the order of two or three L. This is sort of eight times more complex than the LMS algorithm. But remember LMS is sort of an approximate gradient algorithm whereas this algorithm provides the exactly least squares solution after each time step. So it's a reduced complexity compared to the complexity of standards RLS which is quadratic in the filter order. This is linear in the filter order. If you look into the details of each layer, then you will see epsilon signals, which can be identified as forward and backward prediction residuals in linear prediction, forward and backward linear prediction problems as we have mentioned it in chapter 7 as one of the examples or applications of optimal and adaptive filtering, as it would also be seen for instance in speech codec. If you have these forward and backward prediction residuals, they will also be made more explicit in the next slide, multiplying these with product of the cosines will result in true a posteriori or a priori forward and backward residuals.

7.11 Slide 23

In this slide you see the structure of one layer made a bit more explicit. In one layer, as we mentioned already, you're going to see six rotation with six rotation cells. If you go in the slide from right to left, to the far right, you have a rotation cell that represents one element of the boldfaced z right hand side vector, and it’s fed with a signal which is denoted here as a delta signal, and it produces the next such delta signal which is propagated to the next layer. Somewhere in the middle of the slide, you also see the rotation cell that accumulates the product of the cosine of the rotation angles. There the input is product of the cosine up to the previous layer and then you add a multiplication with the cosine of the rotation angle of this layer. Then finally on that is the interesting part in terms of the structure of one layer, you have two intermediate epsilon signals which are denoted here as epsilon with superscript (k-n) and subscript (k-n+1) up to k, and then the other epsilon signal epsilon superscript (k+1) and subscript (k-n+1) to k. These epsilon signals, as it was also indicated, in the previous slide already should be interpreted as residuals from forward and backwards linear prediction problems. If you start with epsilon superscript (k+1), that basically corresponds to a residual of one step forward linear prediction, where you predict the sample at time (k+1) of the input. So you predict u[k+1] from a number of previously observed samples of the signal u, and the time indices of these samples are from (k-n+1) up to k. And similarly, if you go to the other epsilon signal at the input of the layer, epsilon (k-n) is basically a prediction of the input signal sample at (k-n) by making use of samples of the input signal with time indices (k-n+1) up to k. So basically this is a backward prediction where you're predicting an older sample from later samples. So you're using samples from (k-n+1) up to k to predict the previous sample at time (k-n). The operation of the layer then is to take these two epsilon signals, and first to the left, use one of the epsilon signals to compute a rotation angle and then apply this rotation to a rotation cell whose input is the other epsilon signal. And you take the other epsilon signal, feed it into a cell that computes a given rotation, then there's another cell that applies the same rotation which is fed with the other epsilon signal. So you have this crisscross reverse roll type of operation. Here to the left you use one epsilon to computer rotation, the other epsilon to apply the rotation and then to the right you use the other signal to compute the rotation and the first epsilon to apply the rotation. The result of all these operations will be two epsilon signals. In one line you're also going to have a delay operation, which is similar to crisscross operations structures that we have seen in chapter 5 on filter realization. Then the result is two signals where one of which gets delayed and the other doesn't get delayed. You know the outputs of these operations are again two epsilon signals. First look at the epsilon signal with superscript (k+1) and subscript (k-n) up to k. So you're doing a forward prediction of the (k+1) sample of the input signal and now you're using samples from (k–n) up to k. So at

the output you're doing a prediction using samples from (k-n) up to k, where as at the input you have a prediction of the same sample, u[k+1] using previous samples from (k-n+1) up to k. So the order of the prediction is increased if you go from the input layer to the output of the layer, predicting the same sample u[k+1], but using a prediction filter order which is increased by one if you pass one layer. Same thing for the backward prediction. So the prediction of u at time (k-n), using later samples from (k-n+1) up to k at the input of the layer. At the output of the layer, you're going to use previous samples, (k-n) up to k, so this is one additional sample to predict an even older sample at (k-n-1). So again here you see that the backward prediction order has been increased by one. If you take all these epsilon signals and you multiply with the product of the cosines, you get true forward and backward prediction residuals and these are the b for backward and the f for forward, as you see them also in the function of the layer.

7.12 Slide40

Final conclusion slide. If you look into the literature, as I mentioned, there are many fast recursive least squares algorithms available. We've looked at only one of these algorithms, QRD least squares lattice algorithm, which also has good numerical properties because it's basically derived from the QRD RLS algorithm, which in itself has good numerical properties. What you aim is an algorithm with good performance because it's basically still the RLS algorithms, so all RLS algorithms that we have seen up till now starting from the standard RLS to QRD based RLS, and now least squares lattice in infinite precision. When these algorithms are fed with the same input signals, they produce exactly the same results, so they provide you with the solution of a least squares estimation problem at each point in time. So you get high performance at a low cost complexity which is linear in the filter order, similar to LMS. But LMS is a cheaper algorithm and can suffer from very slow convergence. In general, the derivation of these fast algorithms is very mathematical. But if you look into signal flow graphs, the derivation actually only takes few operations in the signal flow graph. And I believe that that indeed leads to additional insights into the derivation as well as into the resulting structure of the algorithm.

8 Chapter 11

Covariance

X hat: $\widehat{\mathbf{X}}$

Boldface:

Transpose Matrix:

orthogonal transformation:

Chapter 11

Slide 1

This is chapter10 on **kalman filters**.

Slide 2

Chapter 10 is the last chapter in part three on optimal and adaptive filter. We have looked into optimal filter design, including the squares filter design from Least Means Squares filter design. We're first going to move into least squares not filter design but parameter estimation. And then as a second step we move from a systematic parameter estimation into dynamic parameter estimation. And see how the common filter offers a tool and a solution to the problem statement there. And then the rest of the chapters will look into common filter algorithms, which will xxx the generalizations of algorithms that we have studied earlier, the recursively scarce algorithm or the scrooge recursively scarce algorithm.

Slide 4

Starting point here is Least squares estimation or squares filter design. As we have introduced it in in chapter nine, the basic formulas are repeated here. Remember that we're designing a filter with filter coefficients in a bold phase w vector, and the filter should be such that when we take an input signal to you, signal and filter it with a designed filter. The output of that should be ultimately close to a desired to filter outputs, and optimally close is defined here in terms of least squares criterion. So, we're summing over the squares of the error samples, the difference between the desired filter output and actually produced filter outputs that involves a matrix. The capital u matrix and a right-hand side, (sort of speak vector), the boldface d vector and then the least square solution, the optimal set of filter coefficients, or the least squares I should say sets of filter coefficients is given here. That is a formula to remember u transpose u as some sort of auto correlation of the input signal matrix u transpose u inverse times u transpose d, which is sort of a cross correlation between the filter input and the desired filter output. So, the formula to remember w squares as u transpose u inverse times u transpose d.

(Transpose Matrix)

Slide 5

Now, the least squares filter design formula from the previous slide is also used in completely different in a sense context. The context of parameter estimation in a linear regression model() as you will see it in this slide, the problem statement is then given as follows. Assume you're given key vectors of input variables or so-called regressors() and these are the bold phase u vectors as we have also seen them in previous slides and then corresponding observations of a dependent variable. So key one of two K. Where the assumption is that these observations of the dependent variable satisfy a linear regression model or an observation model which is indicated here where the observation d if you collect them in into bold phase vector d as we have been doing in the previous slide. Also, that this d is equal to the u matrix constructed with the input variables times an unknown parameter vector which is indicated here as w zero plus observation noise additive observation or measurement, you could say noise. So, the d is equal to u times an unknown w zero plus observation noise or measurement noise. And then the question is if I'm given the u and the d, and I do not know what the w and the e is. Of course, then **the question is what would be a good estimate for the w zero?** And then the question would be if we were to use the formula from the previous slide. So, let's say W least squares this u transpose u inverse **times** u transpose d. The formula to remember is that a good solution to this problem does give us a good estimate for this unknown parameter vector. The w zero. That is the basic question, and the answer will be given in the next slides.

Slide 6

When we use this formula to remember for parameter estimation, we're doing parameter estimation based on a least course criterion and hence this is referred to as the scores' parameter estimation, or the W_{LS} is referred to as the least square estimator. And in this slide and in the next slide, properties of this v squares estimator are provided. The first property is as follows, if the input variables to both ways u vectors are given and fixed. Then the additive noises assume to be random and with zero mean. Then the least squares estimate is set to be unbiased, which means that the expected value of the least squares factor is equal to the w zero. So, if you would do many least squares estimation experiments with different realizations of the of the additive noise vector, then on average you will you would have a result which is equal to the w zero and that is actually easily proven. So expected value as you see it in the formula here expected value of the square solution. Least square solution being equal to u transpose u inverse and u transpose d and the d satisfies the linear regression model as we had it in the previous slide. **Then that leads basically to the expected value of W E squares being equal to w of zero plus something expected value of with an e vector of which the expected value is equal to zero, so hence expected value of the least square solution is indeed equal to the w of zero.** So, this property is referred to as unbiasedness. So, under the assumptions made here the squares estimator or estimate is said to be unbiased. Then under an additional assumption that the noise matrix, the bold phase e has unit covariance().

So, expected value of e times e transpose as an identity matrix. Then the estimation error covariance matrix also takes a very simple form, the estimation error covariance matrix is basically that you take the estimation error which is the Least squares estimate minus the w of zero, that is an error vector, and then you do this vector times its transpose and the expected value there. If you substitute the above formulas, then you end up with a very simple expression for the z covariance matrix which is u transpose u inverse. At this point, this formula doesn't really ring a bell. It's something to remember and it's something that we use further on.

Slide 7

Further to this, we can define the so-called mean squared error of the estimation and it's defined based on the same error vector as we have used it in the previous slide. So, we take our least squares estimate W V squares, we subtract the true parameter vector w zero and this is an error vector and of this error vector we take the squared norm. And then do the expected value zero of. So, this quantity basically says how far our estimate is from the true parameter vector w zero in expected value of the distance as a square to two norms. And you can easily check that actually if you form these 2 norms. This is the sum of elements which are basically the diagonal elements of the error covariance matrix as we have to find it in the previous slides. And some of the diagonal elements is the trace. So, our mean squared error is basically the trace. Or the expected value of the trace of the oracle various matrix which in the previous slide defined to be u transpose u inverse.

Notice that this defined mean square error is different from the another quantity which was given basically the same name, the mean squared error criterion, which was used in chapter 7 to do optimal filter design. What we have here is something with the same name, but then completely different. So, check the formulas there and check the form here and you observe that it's something completely different, but it just so happens that they're given the same name. Now, the mean squared error is obviously something that you would like to have as small as possible. **If you have as small mean squared error, you have a good estimator.** And a nice property then is as follows if you consider a family of so-called Least squares estimators where a least squares' estimator or a member of the family is sort of specified by the function, we will make an estimate of the w, so w-hat(wˆ) as a linear or you could say function of the bold phase d vector. So, a linear estimator w-hat would be capital z times the d plus small vector boldface() vector z where you can think of various sort of ways of selecting the capital z and the and the boldface z. So that is defined to be the class of linear estimators and the Least squares estimator. As we have seen it actually is a member of the family where the capital z and the acute transpose u times u transpose and the boldface small z is equal to zero. So, the v score estimator is a member of the family and then the nice property is that the least scores is estimator actually is the one member of the family that minimizes this defined mean squared error.

So, the least square estimator is referred to as the minimum mean squared error estimator within this class of linear estimators and hence it is referred to as the linear minimum mean squared error estimator. This is all under the assumptions I should stress as we have made them in the previous slide. Under an additional assumption, namely the assumption of the additive noise measurement noise, the e vector being or having a gaussian distribution, it is actually shown that the least estimator is of all possible estimators in the world. Not restricted to the family of linear estimators is the minimum means squared error estimator and that is a very strong statement or property. So, under the assumptions made in terms of the mean squared, error criterion as we have to find it in this slide the least squares estimator is sort of the best estimator you can have. All these properties in this slide and the previous slide make the least squares estimator where the formula in a sense was just borrowed from the completely different theory of optimal filter design and least filter design, but so with this formula we have an estimator which has very remarkable and very desirable properties indeed.

Slide 8

Before we continue, two remarks are in order on this slide and on the next slide, **first remark about the error vector**, the boldface e vector. The previous slide it was assumed that the e vector has zero mean and unit covariance matrix and the

question then could be what happens if the e vector still zero mean but has a non-unit covariance matrix. So, assume the covariance matrix expected value of times e transpose is a more general v matrix with then vs a symmetric matrix. In this case positive definite let's say symmetric matrix. In that case what we're going to be using is so called **Cholesky factorization**. Check your linear algebra courses or Wikipedia on **Cholesky** factor session offer symmetric positive definite matrix. So, this v matrix can always be decomposed into a product of a triangular matrix times its transpose and this triangular matrix is also referred to as the square root and it's similar to ~~(It's basically exactly the same thing as)~~ the square root that we have seen in the previous chapter.The square root can be either an upper triangular or a lower triangular matrix. In this case it's defined as an upper triangular matrix. So, if vs is decomposed into **an upper triangular matrix** TIMES **its transpose**, which is then lower triangular matrix. So, the screw is referred to as feet to the one half. As a square root indeed. This is a quantity that we're going to be using further on. Now, in this case it's demonstrated actually and we're not going to provide any proof here that the linear MC estimator, so of all estimators in this family of linear estimators, the one that achieves the minimum mean squared error, which is again a very desirable property is given, as you see it here in the formula, the w-hat and then an expression. Basically, if you compare it to a previous such expression, instead of u transpose u, you would see u transpose and then times inverse of v, which is plugged in times u and, and, and then the inverse zero. And then similarly rather than u transpose d, you would see u transpose. And then again the v inverse is plugged in and times the t, so that is the formula without any proof here for the linear MMSE estimator and the formula for the corresponding error covariance matrix is also provided here and rather than u transpose u inverse again you would see u transpose and the fee to the minus one gets plugged in times u. And then the inverse there of(zero?). You can easily check that this linear and c estimator actually corresponds to the least squares estimator and a slightly modified observation model or regression model after a so-called pre-whitened operation. So, in your standard observation model d is equal to u times w zero plus the error vector. Now assume that you pre-multiply everything with the inverse **Cholesky** factor. So, times v to the minus one half that gives you a new d vector which is the knowledge here is detailed new u matrix which is denoted here as u tilde (~) and a new error vector which is denoted here as u tilde. You can easily check that for this u tilde expected value of u tilde the times u tilde the transpose is indeed equal to an identity matrix. So, this has become an error vector with a unit covariance matrix and an error vector with the unit covariance matrix is referred to as a wide noise vector and hence this operation is referred to as a pre whitening operation. So, you take your observation model or regression model, you pre-whitened it such that the noise becomes a white noise with the unit covariance matrix and then apply your good alt from the previous form slides the squares estimator to this pre-whitened observation model. So where basically u replaced by u tilde and d is replaced by d tilde and then take the formulas for the w-hat and the corresponding covariance matrix as you see them in this slide.

Slide 9

Final remark here in some estimation problems. Next to the observation with the u matrix and the d vector, you may also have an initial estimate available, which is denoted here as w of zero with a head. And this could be obtained from previous observations or whatever. And assume that this comes with an error covariance matrix which is defined here. So, you take the error vector w add zero minus the exact parameter vector w of zero. This is this is an error vector and then you take this vector times it transpose expected value thereof. Assume there's a given p matrix. Again, this is a positive definite symmetric matrix of which you can do a **Cholesky** factorization. So, assuming you're given the w zero hats and the corresponding error covariance matrix with the p or the p to the one half, then, the question is how do you incorporate this into your estimator? And the solution again without any proof or derivation is as follows, the linear estimator in this case. So again, within the family of linear estimators, the one that provides the minimum **MMSE**. Then the corresponding error covariance matrix, the formulas are provided in this slide. So w-hat, and then you see. Somewhat formidable looking expression and we're not going to go into the details, but basically you can, if you do check the details, you can check that this corresponds to again a least squares estimator for sort of an extended observation or linear regression model with a d vector which is denoted here as d X for extended or something and, and a u matrix which is denoted here as u x sort of

extended again. When you look into this extended observation model, you basically see two parts. The bottom part has the d vector and the u matrix and the pre-whitened operation as we have seen it in the previous slide. And then the top part, if you first sort of ignore the multiplication with p to the minus one half would basically say: identity times the w of zero is equal to the w-hat zero, the initial estimate that you have available, so it responds to basically a part in the u matrix which is an identity matrix identity times the unknown w zero vector would be equal to the observed w zero hat or I should say initial estimate plus an error vector which is the e of zero as you've also seen it in the first formula. And then again you do some sort of pre-whitening operation with the **Cholesky** factor inverse of the **Cholesky** factor of the error covariance matrix. So, again you have and a linear and MMSE estimator that corresponds in itself to a least square estimator of an extended observation or linear regression model as you see it here.

Slide 12

Going to move on to common filters. In the next few slides, you will see that a common filter actually also solves a parameter estimation problem. But whereas in previous slides we looked into a static parameter estimation problem where we're estimating a fixed parameter w zero, we're now going to move into dynamic parameter estimation where the parameter vector evolves over time and the time evolution of the parameter vector will actually be described by means of a so-called state equation and the so-called state-space model, whereas the output equation of the same state-space model actually corresponds to the regression model as you have seen it in in previous slides, a state-space model is then crucial to the common filter and you remember state space models from your signals and systems courses and indeed states models have 2 equations, an output equation and a state equation and sort of state equation will describe the time evolution of the parameter vector and the output equation will correspond to the repression model as we have seen it in previous slides, the details of this will be given in in in later slides. In the next few slides, we're first going to introduce to general common filter problem statement and then slide 14 will indicate how this actually relates to the fixed parameter estimation problem as we have seen it in the previous slide. The common filter is a general tool which is used in many, many applications and it's it is named after Rudolf for Rudy Kálmán, a xxx American electrical engineer and mathematician, who passed away a couple of years ago but basically sort of lives on through the common filter that is used to lead in so many applications.

Slide 13

In chapter 2, we have reviewed time-invariant() time systems or shift invariant time systems, and we have reviewed different ways of describing the input output behavior of such discrete time systems. We looked into impulse response sequences, transfer functions, difference equations and such all to describe are all alternative ways but equivalent ways to describe the input output behavior of such a time invariant discrete time system. The state-based model is yet another alternative way of representing or describing the input output behavior of time in very industry time system and it's provided here by the formula in the middle of the slides. So, the input to the system is still u at time k and the output is still y at time k and the input output behavior is now described through the definition. Basically, you have a state vector which is the boldface X vector at T K and then the state-based model **has basically two equations**. **The first equation** is referred to as the state equation which describes the evolution of the internal state, the acts of k vector and then the output equation. **The second equation** is referred to as the output equation describes how the output signal is add time K a function of the states. The internal state time k together with the input add time k. So, the state equation basically says the next state at time k plus one will be a function of the current state at time k as well as the current input signal u at time k and the function is specified by means of an a and A B, this is a matrix and A B vector. In the case of a single input signal, u of k, so in the case where UK is a scalar input signal.

The second equation describes the output at time k, and so is said that the output at time k is a function of the current internal state at time k. And that together with the input add on K U time k, and this function is again described by two quantity c and A D where sees a. vector in the case where y is a single output and then d is a scalar.

So, we're considering here a single input single output system. So, u of k is a scalar, y of k is a scalar. So, samples of the input signal and the output signal. We can also have multiple input, multiple output systems where the u of k could be a vector of input signals at y of k could be a vector of output signals at k and the ABC and D have corresponding. matching dimensions in any case. So, the state-based model is described by this ABC and D these quantities and is also referred to sometimes as an ABCD model. Notice that if you think of a discrete time system else order system FIR IIR or whatever, then such a system is typically or can be described by means of an elf order difference equation. As we have seen it in chapter 2 and later chapters, what we have here in terms of the state equation is basically rather than an elf order difference equation, we have L first order difference equation where the X times k is a function of X at previous time k and etcetera. So that's a set of L's first order difference equations to indicate that the state-based model is indeed equivalent to other representations of the input, output behavior or descriptions of the input output behavior. Discrete time system, for instance, the last formula indicated here says how the transfer function is related to the ABCD state-based model. And that is a well-known formula from signals and systems courses. So, if you give them the ABC and D, then you can compute the corresponding transfer function of the system.

Slide 14

A very simple example of a state-based model is provided in this slide for an MIIR(?) filter else or IIR filter of which the input output behavior is first described by means of a difference equation with **b** coefficient and **a** coefficient, as we have seen it in chapter 2 and later chapters, we also use direct form realization on this slide of this IIR system with the a and the b coefficients appearing as multipliers in the realization. And you also see a sequence of delay operations in the realization. A straightforward choice for the internal state vector to construct a state-space model is to have the outputs of the delay elements correspond to the individual elements of that state vector. So, what we have here is a fourth-order example with four delay elements and the state vector, the dimension of the state vector should be four if we consider the outputs of the four delay elements then these can be chosen to be the elements of the state vector. So, X one first element that the output of the first delay element X to it, the output of the second delay element etcetera, then the state equation that describes the evolution for the internal state. Basically, to put up the state equation, you have to find out what the state vector at the next point in time at time k plus one will be. Now, if X one is defined at time k to be the output of the first delay element, then the output at the next point in time will be the current input to the delay element. So, the current input to the delay element at time k will be X one at time k plus one as it is indicated by the arrow. And that quantity or that signal can be described as a function of the current components of the state vector X one X two X three X four as well as the input by means of the **a coefficient** etcetera. As you see it in the first row, basically you have the state equation. And then similarly for X two, X two times one will basically be that the input to the second delay element in the delay line and that happens to be equal to X one at time k, so hence the one and then zero, zero, zero in the second row of the state equation. So, in this way you can compile the state equation. And similarly, the output equation, the wire times k can be defined to be a combination of the X components and the input U time k with this state-space model, so that defines ABC and D. You can recompute the transfer function by using the formula on the previous slide, and that should again lead to b of z divided by a of z, or the b and a half the coefficients of the first formula difference equation with the b and a coefficient.

Slide 15

In the filter, we actually going to immediately use more general state space model, which is a state-space model for a time varying discrete time system with additional noise. Time varying means that the ABC and D quantities in the state-space model can actually be time varying. So, vary over time.

Its indices() a at time k, b at time k, c at time k and d at time k, later we're going to assume that ABC and D will be available at each point in time, but so they can vary over time. On top of that, we're going to add a process noise term in the state equation and the measurement noise term in the output equation. So, in the output equation, we say that the output

signal at time k is a function of the internal state of the system as well as the ~~you could say~~ deterministic input signal u at time k, but then there's an additional random term or stochastic() term which is w at time k, which could represent for instance, measurement noise. And similarly in the state equation, we're going to assume that the next state vector is a function of the current state vector Xk plus a function of the so-called ~~let's say~~ deterministic input signal and then plus a random term, unknown process noise term, which is indicated here as v of k. We make statistical assumptions about the v of k process, noise and w of k measurements, noise, we will assume that they are zero mean and white noise sequences and then we also assume we know the covariance matrix of the of the additional noise to v and the w, we're assuming that the v and w are uncorrelated. So, if you do the covariance matrix on the last formula here, you're going to see zeros in the block of diagonal entries because of the assumption of the two being uncorrelated and then noise covariance matrices v of k and w of k, where w of k is actually a scalar v of k is a matrix of which you can again compute **Cholesky** or square root factorization which will be using further on.

Slide 16

So, here's finally the general problem statement for the common filter. The common filter is said to be a state estimator, so it solves a state estimation problem, internal state estimation problem in a system described by state-space model, so we're given a state-based model and given in the sense that we know at each point in time what the quantities the a to b to c and the D R as well as the noise covariance is the V K and the W K and we're also observing the input signal and the output signal to and from the system. So, UK and Y K for all time k and then the question is basically to estimate what is inside the system. So, the internal state vectors add all time k, this is the general program statement and this will be solved by the common filter.

Slide 17

This slide shows how the parameter estimation problem in the linear regression model in the first part of the chapter, is actually a special case of state estimation in a general state space model. If you go back to **slide 4** and check the linear regression model, then you can actually take the linear regression model and substitute it in the output equation, or have it replaced the output equation of the state space model, The linear regression model was given as d of k, the observation at time k is equal to boldface u vector at time k which is the input vector at time k times the parameter vector w zero plus observation noise e of k. If that replaces the output equation then you see that d of k actually takes the place of the y of k the usual output signal from the state space model.

And then somewhat remarkably, the input vector, the boldface u at time k takes the place of the c vector at time k to not be confused here that you is the input vector of the repression model. It takes the place of the c of k and do not be confused with the scalar u of k and the previous state space model, which was referred to sometimes as the deterministic input to the states space model, which actually is not seen in this special instance of the state space model.

So, both phase u of k vector input victory takes the place of the c of k and then e of k the observation, noise takes the place of the observation. Here the output noise Wk the internal state vector Xk has been replaced by the parameter vector w of zero which has received a time index here. So, it's w zero at time k, but then actually the state equation ~~sort of expresses~~ that we assume that this parameter vector is the same at all times K, so the state equation says the parameter vector w zero at time k plus one is equal to the state vector or parameter vector. I should say w zero at time k plus zero plus zero. So, no deterministic input contribution and no process noise contribution. So, we're having or we're adding sort of a constraints to the estimation problem saying that indeed the parameter vector is the same parameter vector at all times k, so this is a special instance of the state space model with special substitutions for basically ABCD and then the noises and the noise covariance etcetera. If you do these substitutions in later algorithms as we're going to derive them, it will turn out that what you derive them with such a substitution is one of the recursively scores algorithms. As we have seen them in in the previous chapter, we will see the standard recursively scores algorithm as a special case. This will be seen in slide 25 and we will see that the Recursive Least Squares (RLS) Algorithm is also a special case of the common scrooge

common filter in slide 22.

Slide 18

To understand later common filter equations and algorithms, we have to introduce a definition and the definition here is a definition of ~~X hat k slash L~~ ($\widehat{X}_{k|L}$) where this X hat is an estimate. It's going to actually be a linear MMSE estimate. Indeed, the comm filter will produce linear minimum mean squared error estimates of the state vector acts at time k and we we're producing this estimate by making use of all data available up until time L, so that means we will be using to produce this estimate of xk we will be using all ABCD matrix all error covariance matrix all input and output samples up until time L.

And with this definition actually we can distinguish between filtering prediction and smoothing as follows: filtering is where the L in the definition is equal to the k, so that means for instance by making use of observations of the ABCD and the error covariance and the inputs and outputs up until time 1000, we make an estimate of the internal state vector at time 1000 prediction is where the L is smaller than the k, so there for instance, we would make an estimate of the internal state vector time 1000 by making use of observations ABCD error covariance input output samples up until time 900 for instance. So, you observe the system up until time 900 and then make a prediction. An estimate of the later state vector acts at time one thousand, smoothing is sort of the other way around where L is larger than k, so you would observe your system up until time 1100 ~~let's say~~. So, you have ABCD error covariance put up put samples up until time 1100 to produce an estimate of the internal state vector at time 1000. So, this is filtering prediction and smoothing. The common filter as we will see it in the next few slides actually computes at time k. The filtered estimate hacks hat k slash k as well as the one step ahead prediction hacks hat k plus one slash k, so we're estimating the internal state vector xk and the next state vector xk+1 by using observations. So, ABCD error covariance inputs outputs up until time k.

For each estimate, we're also going to have a corresponding error covariance matrix, which is defined similarly to error covariance matrix that we have to find in the first part of the chapter. So, if we have an estimate, for instance in the first formula here X hat k slash k, the true internal state vector is xk, so we subtract xk from $\widehat{X}_{k|K}$, this is an error vector. You take the error vector times its transpose the expected value thereof. That is an error covariance matrix for which we will use a similar definition. So, $P_{K|K}$, and again, we're also going to use square root **Cholesky** factor. So, we define $P_{K|K}$ to the one-half times $P_{K|K}$ to the transpose one half is the **Cholesky** factor of $P_{K|K}$ and similarly for the one step ahead, prediction $P_{K+1|K}$ etc.

Slide 20

The state estimation problem that the common filter will consider and solve at time k actually corresponds to a parameter estimation problem in the sense of the first part of the chapter here in a linear regression model, which is indicated here in the formula.

And you will see that in this regression model, the parameter vector that we're estimating is actually sort of a compound vector that has all state vectors from the initial time acts at time zero until at time k plus one, the one step ahead prediction as you will see at time k. What you see in the large formula here is basically the state space models. So, the state equations and the output equations for all time steps from an initial time step zero up to time k, the first equation that you see is a bit of a special equation, it's the inclusion of an initial estimate of the state vector at time zero, sort of the starting point, and this is in the sense of **page 8**, so **you can go back to page 8** to understand the details here. But if you skip this first equation, then the second equation basically reads minus b of zero, u zero is equal to a of zero X zero minus X one is equal to or plus v of zero, and that is a state equation that says basically X one is equal to a of zero times X of zero plus B zero times U zero plus if you have zero state equation at time at time zero. And then the third equation is basically similarly the red were identified as the output equation at time zero.

And on you go. So, every set of two consecutive equations in this large set of equations corresponds to the state space model at a certain point in time. Now the equations are spelled out in a particular fashion so that you can indeed identify

a linear regression model where the vector with all the state vector the x is the vector of unknowns is the parameter vector that will be estimated and then for instance the left-hand side vector is similar to the d vector. Boldface d vector in the standard linear regression model.

The vector of observations whereas the matrix is similar to the u matrix and the standard linear regression model. And then the vector to the far right is the observation noise or measurement noise vector in the linear regression model. Also indeed notice that everything in the left-hand side vector and everything in the large matrix is known, the assumption in the commons filter is that the ABC and D quantities are known, that the input and output samples are known, etc. And so, the left-hand side factor, ~~I should say~~ is known, the large matrix is known. And so, here you have a linear regression model, where the parameter vector is indeed the sort of compound vector with all the state vectors from X of zero to exit k plus one.

Slide 21

A minor point here referring to **slide 7**, is that if in the linear regression model if the observation noise vector (the boldface e vector) has a covariance matrix, which is different from an identity matrix, then you have to include a whitening operation, a pre-whitening operation. This can also easily be included here and **it leads to minor modifications**. You get an impression in this slide but the message is we're actually not going to do this. We're going to keep things simple and compact in later slides. But then if you do have an error covariance matrix() not equal to an identity matrix, you can indeed easily include the pre-whitening operation and have simple sort of minor modifications in later slides to get the appropriate algorithms in this case.

Slide 22

The linear regression model in the previous slides can be seen to have a number of equations in a number of unknowns where the number of equations is given here as capital L plus k plus one times L plus one. Capital L corresponds to the initial estimate for acts of zero, ~~in the sense of~~ **slides 8** ~~remember, and then the~~ cableless(?) one basically refers to state space models that are used from time zero to time k, so k plus one of them. And then the L plus one response to the number of equations in each state space model L for the state equation and then plus one for the one output equation.

The number of unknowns in the linear regression model is k plus one times capital L, capital L is the dimension of one state vector and we have k plus two unknown state vectors that were estimating here from X of zero up to x and k plus one. You can easily check that the number of equations then is always larger than the number of unknowns. So, this is an over-determined set of linear equations and so this is the set of equations that we have to solve and over determined set of linear equations can be solved in at least course sense and as we have seen in the first part of the chapter that indeed leads to a linear MMSE estimate and **this is a desirable estimate** as we have also seen it in the first part of the chapter. So, this is what we're going to be doing if in the previous linear regression model, you identify the u matrix and the d vector if you go 2 slides back, the large matrix is the u matrix and the left-hand side vector is the d vector. Then you can compute this linear MMSE estimates by using the formula to remember from the beginning of this chapter. So, the estimate would be u transpose u inverted times u transpose d and that would lead to an estimate of all state vectors from state vector X zero up to state vector X K plus one, and to produce these estimates, notice that we have used observations up until time k, so **the end result of that linear MMSE estimation would be X hat zero slash k**, so the estimate of X of zero using observations up until time k and then X hat one slash k etcetera up until X hat k plus one slash k sort of one step ahead prediction of state vector X at k plus one using observations up until time k.

Now using this formula to remember u transpose u inverse times u transpose d, as you may remember from the previous chapter **is not a very good way of computing**. The square solution also in the recursive algorithm, this does not lead to a good algorithm. So, we're going to do something different and something different will again be based on QR factorization and back substitution as we have seen it in the in the previous chapter. And that should also lead to a recursive algorithm.

Again, screwed algorithm based on QR factorization and back substitution. It will be a bit different from what we have seen in the previous chapter because notice that in this linear regression model, at each time k, the number of equations in the linear regression model **grows**, but also the number of unknowns grows larger as time increases because we're estimating more state vectors. So, even more so here **we need a recursive algorithm to sort of keep complexity at a reasonable level** and these recursive algorithms will be the topic of the next few slides.

Slide 23

A recursive implementation will rely on QR factorization or QR updating and back substitutions, similar to what we have seen and studied in chapter 9 to develop a square root recursive algorithm and this will lead to recursive square xx filter algorithm here.

From chapter 9, we remember that in such an algorithm, from time step k minus one to time k, basically a triangular matrix is propagated together with a right-hand side vector and from this triangular matrix and right-hand side vector, basically at time k minus one, the squares solution could be computed so this is the shaded or colored part in the first equation. So, a triangular matrix and a corresponding right-hand side vector at time k. New observations are added to the set of equations and this basically corresponds to the state based model at time k.

State-based model that has the ABCD matrix at time k as well as the input and output samples at time k the uk and the yf. Notice also that we have not only added new observations, so added rows to the matrix to the left in the equation. But we've also added an unknown state vector x k plus one, so we've **increased the number of unknowns** in a set of equations and that leads to the additional zero to the right of the colored triangular matrix, and then the i in the same column.

So, we've increased the number of equations as well as the number of unknowns. And now the question is, so solve this set of equations. Again, this is an over determined set of equations and all the unknown x. To do this, we should first turn the matrix to the left, into a triangular matrix, and we do this by means of orthogonal transformations(). And then we apply the same orthogonal transformations to the right-hand side vector, and that will lead to a new triangular set of equations on the right-hand side from which we can compute. So, the next estimates of the all the state vectors. Now it turns out if you remember the details of the sort of the update process to turn triangular matrix with an additional, or in this case, several additional rows into a triangular matrix, again with a sequence of given transformations where **you basically introduce zeros in in a newly added row from left to right**. Here, you would see that in the newly added rows with the Ak and the Ck in the matrix, basically you have already a number of zeros to the left, so when you start applying/giving transformations to turn that matrix into a triangular matrix again. The first so many orthogonal given transformations are basically going to be void transformations **that basically do nothing**. And based on that, the observation is that in order to turn that large matrix in the first equation into a triangular matrix, basically only the lower right part of that triangular matrix is going to be active. And basically, together with the new part with the A and the C and the minus i, that smaller matrix has to be turned into a triangular matrix. And so, in the meantime, when you do that by means of orthogonal transformations, you apply the same orthogonal transformations to the right-hand side vector. And there also you're going to see that only the lower part of the right-hand side vector will actually be used for the QR updating process. And then remember, after the QR updating, you would do a back substitution to compute the vector of unknowns, in this case the vector unknowns, has all state vectors from time zero X of zero up to time k plus one at time k plus one, but actually in a commons filter, only the last two stage vectors are actually computed at time k, so at time k we're going to compute X hat K slash K and X hat k plus one slash k as it was indicated in in **slide 15**. And, and the final observation here is when you do back substitution, you indeed start computing the vector of unknowns from bottom to top. And, so basically if you have to compute only the bottom part of the vector of on unknown, then you can sort of **abort the triangular back substitution process after you have computed indeed x hat k plus one slash k and x hat k slash k** and then basically so you stop the back substitution part. That actually means that only the framed parts of the set of equations both in the

left-hand side matrix as well as in the right-hand side vector is the active card that that will be used in the update process at time k and everything else from the larger triangular matrix and all previous components in the right-hand side vector will basically remain untouched and are not needed and so do not have to be remembered. So, the conclusion from this slide, and this is an important conclusion, is that only the framed part in the matrix and the framed part in the right-hand side vector will be active in the update at time k and this is basically the thing that we continue with the next slide.

Slide 24

In the previous slide we conclude that the relevant sub problem at time k is a much smaller problem where in the first place we have a triangular matrix which is propagated from time k minus one to time k which is the colored triangular matrix and then a corresponding right-hand side vector triangular matrix is L by L, and the vectors also has L components and then add time k, we add basically the state equations at time k with ABC and D at time k and then input sample you can output sample yk, and this altogether forms a set of linear equations.

Again, it's going to be an over determined set of linear equations from which we're going to compute basically an estimate for the state vector at time K X K and the state vector, the next state vector at time k plus one, so we're going to estimate or compute X hat k slash k and X hat k plus one slash k because we're using observations up until time k.

The triangular matrix that is propagated from time k minus one to time k can actually also be given a name. ~~In a sense~~, remember that this triangular matrix in the previous time step was the lower right part of larger triangular matrix from which by back substitution and estimate is computed again for the state vector at time k, but at that point in time using observations up until time k minus one, so that triangular matrix, the colored triangular matrix here in the corresponding right-hand side vector was used in the previous time step to compute X hat k slash k minus one, so if the in the triangular matrix is a triangular matrix, then the right-hand side vector is this triangular matrix times X hat K slash K minus one so that if you do the back substitution you indeed end up with X hat time K minus slash k minus one. Now the triangular matrix if you go back to slides 8 and 5, you would see that this matrix times its transpose and then the inverse actually defines an error covariance matrix. For this estimate, X hat k slash k minus one, so this is eventually seen to be the inverse square root of the error covariance matrix for state vector k slash k minus one.

So, this is the PK K minus one to the minus one half. So, we have identified the triangular matrix to be this expression basically, and then in the right-hand side vector but see the same p to the minus one-half times the X hat k slash k minus one.

Slide 25

The resulting recursive algorithm, which will be referred to as a square root common filter algorithm, is actually a very simple one and is expressed by means of the following two formulas, so for each time. K running from k is equal to zero to k is infinity. We're going to execute the following two formulas. And the first formula basically is a QR update formula if you start to the to the right you propagate a triangular matrix from time k minus one to time k together with a corresponding right-hand side vector. This is the color part, and in the previous slide we have identified the triangular matrix as the square, inverse square root of an xx matrix, etcetera.

And then in time k, you append the state equations for a time k with the ABCD matrix and the samples of the input signal uk and output sample yk and then you apply basically an orthogonal transformation, the q matrix, to triangularine the part of that so-called pre array matrix, the part of the matrix to the, to the left of the vertical line. So, everything that is to the left of the vertical line is then triangularine **leads to a twice as large triangular matrix**. So, if the initial color or the triangular matrix is L by L, so the new triangular matrix is now two L by two L. And the same transformations are obviously applied to the right-hand side vector. That leads to a larger triangular matrix of which then again, the lower right part is selected together with the corresponding part in the right-hand side vector and there you can do a back substitution to compute X hat k plus one slash k prediction, one step ahead prediction of the state vector at time k plus one and then

the colored triangular matrix and corresponding right hand side vector is propagated from time k to the next time k plus one, and so this is situated over time in each time. Step you basically inherit a triangular matrix and a corresponding right-hand side. You do a QR factorization of basically twice the size and then propagate again a part of the resulting triangular matrix and corresponding right-hand side.

Slide 27

In slide 14, it was explained how the fixed or static parameter estimation problem from the first part of this chapte can actually be viewed as a state estimation problem with a state-space model with specific or special substitutions. It turns out that i if you use these specific or special substitutions in a square root column filter, then you obtain an algorithm which is exactly equal to the square root, which are less algorithm that we have derived in chapter 9 and this is indicated in in in this slide.

There's one tricky part to the substitution here, which is the colored part with V K to the minus one half and minus V K to the minus one-half root basically for the special substitution the. And that actually corresponds to the process noise variance being equal to zero and then V K to the minus one half is actually, in a sense, infinitely large. And so, the colored part actually corresponds to sort of a constraints that says that the first half of the vector of unknowns is equal to the second half of the vector of unknowns.

And so that basically or essentially you can drop half of the number of unknowns, which means that in a compound pre array compound matrix, the pre array, you can basically remove the second column and that leads to the final formula as you have it here. So, the substitution is a bit tricky, but the r~~emarkable part,~~ remarkable conclusion is that indeed the squared root or less algorithm from previous chapter is a special case of a squared root column filter algorithm. As we have it here, sort of square root common filter. The other way around generalizes the square root **RLS algorithm** as we have seen it in, in the previous chapter.

Slide 29

If you go to the library and check the textbooks, then remarkably this square root common filter algorithm from the previous slides is hardly ever given. And what you would mostly see as an algorithm, which is referred to as the standard common filter algorithm or the conventional common filter algorithm and which is also given here in this slide, we're not going to go into great detail here, only saying that after an initialization step, the standard comment filter basically has two update equation. **First, a measurement update equation, and then a time update equation and the measurement update**. Basically, if you look to the second formula, you would compute X hat K slash K is equal to X hat k slash k minus one plus sort of a correction term and the meaning of that formula is basically that X hat k slash k minus one is the one step ahead prediction of the state vector at time k computed at the previous update at time k minus one and with the additional correction term that is then turned into a filtered estimate of X K, so X hat k computer that time k, so X hat k slash k. Where the time update and I should also say the measurement update sort of corresponds to the output equation of the state space model. That has to see in the d matrix or vectors let's say, and you see them indeed appearing in these measurement update equations.

The time update equation then corresponds to the state equation and the state space model. And they basically from the filtered estimate, X hat k slash k. You compute one step ahead prediction of the next state vector at time k plus one, so X hat k plus one, slash k also computed at time k. Each estimate also comes with an error covariance matrix or the first formula for the measurement update. And the first formula for the time update indeed provides the corresponding error covariance matrices here. You can also already see that if you look into the measurement update that the equations somehow smell like the equations of the standard recursively squares algorithm that we have derived in chapter 8, and indeed in two slides from now we will say that the standards are less algorithm is in fact a special case of this standard comm filter algorithm.

Slide 30

We will not go into further detail of the equations or formulas in this standard common filter algorithm, but an obvious question could be is this actually the same thing as the square root common filter algorithm? As we have seen it, and **the answer is indeed yes**, one can easily be derived from the other. So, for instance, if you start from the square root common filter update equation. Then at time k, remember we basically solving an over determined set of linear equations and two unknown state vectors. And this over determined set of linear equations can basically be sort of split into an over determined set of equations in X hat k|k, if you consider in the matrix to the left here in the equation, if you consider the first row with the p to the minus one half together with the last row with the ck, that is basically L plus one equations only in the unknown X hat k slash k.

And if you use a closed form solution for this. Basically, for solving this X hat k slash k from this over determined set of equations, basically using that formula to remember, remember that u transpose u inverse times u transpose d, then you would that would result in a measurement update as you have it in the previous slide. And then the rest of the over determined set of linear equations the second row and the matrix to the left with AK and the minus one. Minus identity is basically once you know what the X hat k slash k is basically L equations to compute X hat k plus one slash k from X hat K K and that corresponds to the state update equation or equations as you had in the, in the previous slide. So, conclusion is there's all you can very easily, derive the expressions, formulas from the previous slide, from the formula as you have it in and in this slide also the corresponding error covariance matrix, update formulas, ET cetera. That is rather simple. So, they do indeed represent the same procedure and an infinite precision. These two procedures would indeed provide exactly the same outputs, but then similar to the conclusion and in the previous chapter in finite precision, it is advised to use the square root algorithm which is better behaved numerically. So hence the square root algorithm is preferred over the standard common filter, which remarkably is given in all textbooks.

Slide 32

And then the final slide or comment here. If you again, go back to slide 14, it was indicated through the fixed or static parameter estimation problem from the first-half of that chapter corresponds to a state estimation problem with a specific substitution in the state space model. Again, if you use this specific substitution in the formulas of the **standard common filter of slide 23**, then what you obtain is indicated in in this slide were basically the measurement update equations indeed correspond to exactly the update equations of the recursive standard recursively squares algorithm that we have seen in chapter 8, whereas the time update equations are basically unnecessary or avoid equations that you can in a sense skip or you just give different names to the same quantities. So again, you could say that **the standard RLS algorithm is a special case of the standards column filter** from the previous slide or vice versa, that the standard common filter generalizes the standard recursively squares algorithm from chapter 8.

9 Chapter 12

Chapter 12

Slide 4

To develop the basic filter banks set up in the next few slides. The starting point is the graph indicates in in this slide and it refers to a particular strategy for processing a signal which is referred to as sub band processing. In later slides, we're going to see applications of this where the processing we have in mind is an equal cancellation, for instance, and adapt to filtering context or in audio coding and decoding application, and you could envisage a general processing, which could be difficult to do on the as we call it full band signal on the input signal as a whole.

And the strategy is then to split the input signal into a number of subband signals by means of a collection of filters and the graph it's, the H of zero, h one, h two, h three. and so, this is a small example where the number of channels is Only 4, but in practice, the number of channels can be a much larger number of thousands, let's say. So the idea or the strategy is to split the input signal into subband signals. So, in the example, four of them here, and then do individual processing of each such subband signal. and so in the subband we could do an encoding decoding of the audio signal debts sits in in a particular subband or we could do an acoustic equal cancellation by means of an adapt to filter applied to the signals in a particular subband, et cetera.

So For this where we will need a collection of filters, and as you see it here the h of zero, h one, h two, h three, our collection of filters that or each filter that selects a portion of the full frequency range Hof zero in the graph here. It seemed to be a low pass filter, and so it selects the low frequency content of the input signal, and then h one selects in a different frequency band and h two yet another frequency band, three, yet another frequency band.

And altogether, obviously, the filters are assumed to expand the full frequency range. So, we apply a set of filters to the input signal. Then we have subband signals or signals which are referred to as subband signals. We apply a processing to the individual sub band signals. This is referred to a sub band processing. And then the results of these subband processing operations were somehow combined into an overall output signal, which is then a full band, full range full frequency range, output signal again. So, these are the basic operations that we have in mind. We'll see a few applications in later slides. But an important aspect will be that in order to implement such a scheme in an efficient manner who will have to implement it as a multi rate structure with up and down sampling operations as you will see it in the next slides.

Slide 5

Of the slide. And in the next 3 slides, we will develop the basic filter banks set up step by step. And this is step one. It’s the filter bank as we have seen it already in the previous slide. And this filter bank will be referred to as the analysis filter bank. And in a sense, it's analyzes the sort of frequency. Contents of the input signals. So the analysis filter bank is a collection of filters. Capital n will denot, the number of filters and the analysis filter bank. And all filters are fed with the same input signal. So you have your input signal and that runs into the all of the filters and the analysis filter bank, as we added in the previous light, the Hof zero, h one, h two, h three, the analysis filters will also be referred to as decimation filters.

And the reason for this will become appearance in in the next slide actually. Ideally, you would have filters with an ideal frequency response, namely an ideal band pass or low pass or high pass or whatever response. But ideal in a sense that it's a uniformly flat response, in a pass band and then an infinitely steep transition Anda zero response and stop band so if all of the filters are ideal filters as you have it here in the in the first graph, then you could speak of an Ideal analysis filter back.

Such ideal bad best filters are also sometimes referred to as ideal brick wall filters. In practice, of course you're going to see filters which are not brick wall filters which will have ripples in the bath bands and ripples in their stop bands and not infinitely steep transitions. And that will lead to specific effects with which will have to be analyzed in a filter bank set up.

Slide 6

Step 2 is a decimation on or down sampling after the analysis filter back. So, by means of decimators or down samplers. The reason for having this is basically to increase efficiency. If you have the simple example as we have it here with 4 analysis filters, then obviously, if the input signal has a certain number of samples per second due to particular sampling rate, then with 4 analysis filters, we're going to have 4 times that number of samples per second, which then will have to be processed by in the sub band processing.

This is a simple example, but think of a large analysis filter bank with 1,000 analysis filters that would give us 1,000 times more samples in all of the sub band signals compared to the number of samples and the input signal.

So, then in order to reduce the number of or the total number of samples that we have to process in the sub band we're going to apply a down sampling operation to each and every subband signal, the Nyquist sampling theorem actually also tells us that this is something which is allowed because we're reducing the frequency range of a signal. And hence, we can indeed also **reduce** the sampling frequency for that signal. So we'll have the summation or down sampling after the analysis filters. And the decimation factor will be denoted here as the capital d and then the question is what are good settings for this capital d an obvious setting is where the capital d is equal to n so the down sampling factor is equal to the number of filters in the analysis filter bank.

So in our simple example, you would have 4 analysis filters and then 4 fault down sampling after each analysis filter. For that case, the number of the total number of sub band samples, so samples for all of the sub band signals together is equal to the number of full band samples. So, samples of the input signal. And that may sound like maximum efficiency. But then at the same time, we have to or we could be worried about aliasing effects after non ideal analysis filters as we will analyze it.

Further up in the chapter, another option is to use a down sampling factor d which is smaller than the number of analysis filter, so **d is smaller than n** then the total number of sub band samples is going to be larger than the number of full band samples. But in terms of aliasing, this may be a bit eruption in in that it may reduce aliasing effects in in the sub band signal. As we're also going to analyze it further up in the chapter. Notice also that um the analysis filters now proceed a down sampling operation or decimation operation and hence could also be viewed as decimation or aliasing filters, which are there to avoid aliasing and the subband signals after the decimation. So, the analysis filters were also referred to as Decima on filters preceding the down sampling operation.

Slide 7

Step three, then the actual subband processing. And this is the main operation basically because this is the operation for which we have designed to, the analysis, filter bank. And somewhat remarkably, actually, it will turn out that in most of the analysis that we're going to see in the simulator chapters. We're going to assume that the subband processing is basically not there in that sub end processing leads to subband output signal, which is equal to the subband input signals. We will see a very explosive example of this in terms of audio coding.

So, imagine the input signal is as an audio signal. You split it into frequency bands, sub bands, and down sample the sub band signals. And then each down samples sub band signal would be encoded. So represented with fewer bits per second. And then the coded or encoded sub band signals would be either transmitted over a channel, for instance, communication channel or restored into, let's say, a computer memory for later usage.

And then indeed later if you want to read out listen to this audio file, you would read what is stored in the computer memory and then decode the bits that you read from this this memory to reconstruct the sub and signals and then from their reconstruct the full bands audio signal, so in this case. And we'll return to this in a later slide, the sub band processing, which would include the encoding operation and then the transmission or storage. And finally decoding operation. And in the case, of a lossless coding and decoding operation, indeed, the output signals from the sub band processing would be exactly equal to the input signals to the subband processing. So, in that case, indeed, you can assume that subband processing is not there. And so again, remarkable e this is what we're going to assume and later analysis that the subband processing, which is actually the processing for which we indeed designs the analysis. And then later the synthesis filter bank. Nevertheless, we're going to assume that the sub band processing is basically not there.

Slide 8

Finally, from the process sub band signals, the idea would be to construct or synthesize the output signal, the overall output signal. and This is done here in this slide and represented as step four and five, where you will see s step four, basically the expanders or up samplers implementing an expansion or up sampling operation. And then step five will be

actually yet another collection of filters which will be referred to as the synthesis filter bank. First of course we will have to restore the original at sampling rate, so in the simple example as we have the input signal which is filtered by the analysis filter bank. And then we have applied a threefold down sampling operation, obviously, to synthesize the output signal. We first have to return to the original sampling rate to apply a threefold up sampling operation.

Now, if you apply it a threefold up sampling operation, go back to review chapter two. This is where you insert two zeros between every two consecutive samples of the process subband signals. so whenever you introduce zeros, you would have to have this up sampling operation followed by and interpret a filter or filtering operation which is somehow there to interpolate the zeros or filled at zeros with meaningful samples. And this is where indeed we have the another set of filters. and so, we have an f of zero and f of one and f of two and f of three. These are referred to as interpolation filters or synthesis filters that synthesize the overall output. Signals. So the outputs from the synthesis filters are summed, and that summation leads to the overall output signals.

The question then is what should the frequency response as of the centre filter is the f zero, f one, f two, f three in this simple example, what should the frequency response as look like? And It will turn out that basically or basic statement will be that the frequency responses of the synthesis will be similar to the frequency response of the analysis. in a sense that if for instance, age of zero is a low pass filter, then h and then f of zero will also be a low pass filter.

And If h one is a particular band pass filter, then f one will be a similar band pass filter, banning a similar frequency range. We're not saying that the analysis filters and the synthesis filters have to be equal. We're not saying that f of zero should be equal to H of zero and f one should be equal to h one. We're only saying that they have to be similar or related and the details of this statement will become um clearer in in later slides.

Slide 9

Summarizing from the previous slides, this is the picture to be kept in mind. We have the overall input signal which is first filtered by the analysis filter bank which is a collection of filter analysis filter as that also actors anti-aliasing or decimation filters proceeding the down sampling or decimation operation.

Remember that the number of channels in the filter bank, the number of filters in the analysis filter bank is the capital n in the example n is equal to four here. And the down sampling operation will always be represented by d where in this example d is equal to three.

Then after the destination or down sampling operation, we have the actual subband processing.

And the result from the subband processing, the processed subband signals then run into up sampling operations or expansion operations to restore the original samples rate. And after the up sampling operations will have to have a synthesis filters in a synthesis filter bank and where the synthesis filter are also referred to as inter relation filters. And then finally the results from these synthesis filter as the output signals from these synthesis filter are summed into the overall full band, again, output signal.

Slide 10

Now a crucial concept in filter bank design will be the concept of perfect to destruction and the meaning of a perfect structure and is as follows. We have already indicated in a previous slide that we'll often assume that the subband processing does not modify the subband signals. So, in terms of the subband processing, the output signals would be equal to the input signals. And that makes sense, as we will see in the next slide. For instance, in audio coding, the input signal, overall input signal would be an audio signal, which is then split into subband, audio signals, and then each sub band signal would be encoded. And then stored in a computer memory, for instance, read out and decoded. and if that is an ideal operation, then indeed, the decoded sub band signals would be equal to the encoding or the sub band signals input to the encoding operation.

so admittedly somewhat strange. If we assume that the sub band processing is not there, then of course, still referring to that audio coding and decoding application, you would hope that the overall full bands audio signal at the outputs of the synthesis filter bank is equal to the full band audio signal at the input to the whole operation up to, for instance, at most, processing delay, which does not in a sense, degrade your signal.

So the question would then me if the sub and processing is ideal or would not be there so if the subband processing output signals would be equal to the subband processing input signals with the overall output signal to that to the far right of that the scheme be equal to the input signal to the far left of the scheme up to add most delay.

And The thing that is somehow worrying are are the basically to down sampling operations because we have Sina, we know that a down sampling operation after a normal ideal, this emission filtering operation in the analysis filter back. The down sampling after non ideal decimation filtering will introduce aliasing effects. And we have learned that aliasing introduces a distortion to signals that destroys the Information in a signal and can never be sort of corrected. And this is what we're going to have here. So We're going to have non ideal analysis filters and the analysis filter bank followed by down sampling operations. And the down sampling operations are going to introduce aliasing effects. And sort of main question here is, in spite of all of the aliasing effects, can you still have perfect reconstruction, which means that the output signal to the far right of the scheme would be equal to the input signal to the far left of the scheme up to at most delay. And this is a problem statements that will be tackled further up in the chapter.

Slide 13

Before continuing, let us mention two filter bank applications. There's obviously many more. So, filter banks are used in many applications, but here's only two applications, one in this slide, one in the next slide. In this slide, we mentioned sub band coding, which also has been mentioned in in previous slides. In general, coding, the ideas to start from a signal, which can be an image, let's say, or audio signal or whichever signal. And then the ideas to represent that signal with as few bits as possible per second without damaging basically the quality of the signal.

So, think of an audio signal, for instance, which has originally sampled at, let us say, 48,000 samples per second and for each sample where each sample is represented by 24 bits, let's say that leads to an initial number of bits per second to represent the audio signal.

And so, the idea would be or the goal would be to reduce the number of bits you need to represent this the same audio signal. So fewer bits per second without affecting or damaging the quality of the audio signal. And so, the procedure, as it has been indicated already in previous lights is to take the initial audio signal or other signal, split it into sub band signals and apply down sampling. So, this is the analysis filter, bank operation and the summation operation. And then you would um and called each of the sub band signals separately. And this is where you need an encoding strategy. And a simple encoding strategy would basically be to use many bits for a sub band signal that has lots of energy or power. And whereas you would use only few bits for sub band signals that have only little energy or power. That's a very simple strategy that you could buy here to represent the full band signals and then full band that the encoded sub band signals could be stored in a computer memory or on a CD or whatever.

And then read out if you if you want to decode the original signal. So, the decoding process would be to first reconstruct to sub band signals and with the decoding operation. Soto to in a sense, invert the sub bands encoding operation, and then you would just reconstruct the full band signals by means of the synthesis filter bank following the expansion or up sampling operation to undo the down sampling operation at the beginning.

And so, this general scheme could be applied to images for image coding and sub ant image coding is really popular. Some of the filter banks which are used her or for instance wavelet filter banks and we will return to this.

And in a later chapter actually and sub band coding is also used for audio coding to encode audio signals, most of the coding formats which are currently used are indeed sub band coding schemes and further to the simple sub band encoding

strategy that was indicated earlier up nearly energy based encoding strategy for audio coding, especially um the strategy will also try to exploit perceptual properties of human hearing. And this is where or this what is referred to as perceptual audio coding where you exploit masking properties, et cetera, but will not go into the details of this year.

Slide 14

Second example, which is mentioned here is sub ban adapt to filtering where adaptive filtering as adapted filtering as we have seen it in previous chapters. And one application of adaptive filtering, the one which has highlighted here is acoustic echo cancellation. In acoustic echo cancellation, the basic scheme is repeated here. We use an adapt to filter to model an acoustic path between a loudspeaker and a microphone. So, the adaptive filter basically uses a linear model, linear transferred function to model the coupling in between a loudspeaker and a microphone. and even though conceptionally, this is very simple. It’s the basic sort of application for adaptive filtering that i've often highlighted. In practice, this can be quite difficult.

And the difficulty is that the system between the loudspeaker and the microphone, typically has a very long, impulse response, I think, involves responses can be very long and have hundreds or even thousands of samples and are very time varying.

So, if you have um evaluate an acoustic transfer function at this point in time, and then 2 seconds later, you could see a completely different acoustic impulse response. So even though conceptionally, acoustic echo cancellation is very simple and practice, it can be very difficult. And then the idea could be to through the modeling exercise where you model the coupling between the loudspeaker and the microphone in sub band. So, you split signals into sub band and then half the modeling exercise for only short portions or small portions of the frequency range, as you see it in individual sub bands. And that is indicated in the next slide.

The overall sub band adaptive filtering scheme would then look something like this. First, we would have the analysis filter bank and the down sampling. In this case, this analysis filtering and down sampling is actually applied to two signals. So, it will be applied to the loudspeaker signal. On the one hand, this is the top part in in in the graph. So we have a simple, yeah in this case, analysis filter bank again with four analysis filters, H of zero, h one, h two, h three noticed that in in practice, the number of four would not be four would be 1,000 or 2,000 or whatever.

But to keep things simple, we have a 4 channels filter bank here with, let's say threefold down sampling again applied to the loudspeaker signal and then also applied to the microphone signal. So, the microphone signal also runs into an analysis filter bank with the same filters h of zero, h one, h two, h three. And again, the same down sampling operation. Now think of the analysis filter bank where H of zero, for instance, takes out the low frequency content of the signal. So H of zero would be a low pass filter. Then the adaptive filter in the graph here. The adapt to filter in the top branch would basically model the acoustic echo path from the loudspeaker to the microphone in the low frequency range. So, the first adaptive filter, what which sort of acts between the output of the H of zero analysis, filter fed with a loudspeaker signal, together with the h of zero analysis filter output when it's fed with the microphone signal, this adaptive filter is basically there to model the acoustic echo path from the loudspeaker to the microphone, but then only and for the low frequency range.

And that in itself you could assume could be a simpler, an easier task then um using an adapt filter to all of, the acoustic coupling between a loudspeaker and the microphone for all frequencies at once.

At the same time so here, you're focusing only on a smaller frequency range, and hence the modeling exercise with this adaptive filter could be and easier modeling exercise.

Similarly, you would have an adaptive filter acting on the outputs of the h one analysis of filters and an adaptive filter acting on the outputs of the h two analysis filters, and similarly or finally and adapt to filter acting on the outputs of the H of three analysis filters after the adaptive filtering, which is basically there to remove any echo contribution from the sub

band microphone signals. You reconstruct an output, a signal by means of an up sampling and then a sense synthesis filter bank. Notice that perfect reconstruction is also an important concept in this system in a sense that you can view the microphone signal as having two components two signal components. On the one hand, it may have a desired near and speech signal. So, the speech signal of the speaker in front of the microphone and then on some of that near and speech signal, it will have the microphone signal will have an undesired echo contribution.

Now if you run from left to right in the scheme, this microphone signal first is analyzed by means of the analysis filtered band pass sampled. And then you have to adapt a filter operations where basically the or hopefully you could say the echo contribution is taken out. So after the summation operations, as you see them in the slide here, the hope is that the echo contribution has gone and only the near end speech signal remains in the sub band signals. So, in in in a sense, you could say the desired part of the microphone signal uh is analyzed by the analysis filter bank down sampled survives the operations with the adapter filter and then gets up sampled and then filtered by the uh synthesis filters to produce the overall output signal.

And the hope is of course that the output signal is exactly equal to the near end signal. And that can only happen if indeed the analysis and the synthesis filter bank form of a perfect reconstruction filter back so that indeed, as far as the neuron uh desired speech signal is concerned, from input to output from left to right in the scheme, you don't have you're not introducing any filtering or distortion by means of the analysis and synthesis, filtering and up and down sapling operations. So here again, you see that perfect reconstruction is indeed an important concept or aspect.

Slide 16

In the next ten slides, the ideal filter bank operation is illustrated to develop a bit of a feel for a filter bank operation so, but first in the easy ideal case with ideal filters, ideal analysis and synthesis filters and also by means of a very small and simple example with four-channel filter bank. So, four analysis filters, four synthesis filters and maximal decimation, where the decimation factor, the down sampling and up sampling factors are equal to the number of channels or filters in the analysis and in the synthesis filter bank. So in this simple case, number of filters in the analysis and synthesis filter bank is four and the down sampling factory and as well as the up sampling factor is equal to again four. You can also consider slightly more advanced or other cases where we have an over-sampled filter bank where the D is smaller than four etc. These are sort of simple variations on the same theme. But this is the simplest case you can think of and this is where you can analyze the ideal filter bank operation.

So that the filter bank is said to be ideal when it has ideal analysis and synthesis filters and the analysis filters are given here or at least the spectra of the analysis filters are given here in the first graph. And actually, the synthesis filters will be exactly equal to the analysis filters as you will see it in later slides. So, F0 will be equal to H0, F1 will be equal to H1 etc. Now all of the filters in the analysis filter bank are ideal filters with a uniformly flat response in their pass bands and infinitely steep transition, and finally uniform equal to zero response in the stop band. So, brick wall filters as you see it in in the first graph. H0 is seen to be a low pass filter with a pass band from -1/4pi to +1/4pi, so H0 is seemed to be symmetric about zero Hertz or DC and hence, a side remark, it will have a real valued impulse response. Then H1 H2 and H3 are shifted versions of this H0. So, H1 is a shifted version of H0 where the pass band from H1 starts at 1/4pi and ends at 1/4pi + 1/2pi, the bandwidth of each of the analysis filters. Notice that H1 unlike H0 is not symmetric about zero Hertz, so it's going to have an impulse response which is not going to be real value but will be complex valued. We'll see examples of this in later slides. And similarly, H2 and H3 are again shifted versions of H0. Again, not symmetric about DC, hence with complex valued impulse responses. So, if this is our ideal analysis filter bank, so with four ideal analysis filters, brick wall filters, then we're going to apply the filter bank operations to an input signal with a spectrum, sort of a randomly chosen spectrum as it is indicated here. This is supposed to be a periodic spectrum with a period of 2pi. And so this is a spectrum of the input signal. Let us now trace the spectra of other sort of intermediate signals in this filter bank operation.

Slide 17

Let us now trace signals in the top branch and the filter bank scheme. The branch with H0 and F0, the low pass filter. So, H0 first is an ideal low pass filter. If you input a signal with the spectrum as we had in the previous slide, the red spectrum, then the spectrum of the output signal from the H0 low pass filter indicated here as x1, which is simply a low pass version of the red spectrum of the previous slide. So, H0 is our first ideal low pass analysis filter and it will also act as an anti-aliasing filter preceding the four-fold down sampling.

Slide 18

And the down sampling operation leads to a signal which is indicated here is x1 prime of which the spectrum is, also indicated here, a four-fold stretched version of the spectrum of x1 as you have seen it in the previous slide. So, the red spectrum is a four-fold stretched expanded version of the spectrum in the previous slide. But then also images or repetitions of the spectrum are added to turn that spectrum again into a spectrum with a period of 2pi. So, the stretched spectrum first has a period which is equal to 2pi*4=8pi, but then because of the repetitions of that spectrum, which are added, the period of two by is restored. The first such repetition is indicated here, it's the green part as you see it in the graph here.

Also notice that we're assuming that the sub processing is not there or does not change the sub and signals. So, the output from the sub band processing is again the same signal x1 prime with the same spectrum.

Slide 19

Then we have the up-sampling operation, four-fold up sampling operation which leads to a signal indicated here as x1 prime prime. And the spectrum of x1 prime prime is basically the spectrum of x1 prime as we had in the previous slide, which has been four times compressed. So, it's a four-fold compressed version where you basically take the signal over range from 0 to 8pi and you can press it into the spectrum that runs from 0 to 2pi. So, this is where you see the original in red spectrum, which was basically the output of the H0 analysis filter, and then the green images or repetitions of that red spectrum which are added to the red spectrum.

Slide 20

And then the final operation in the top branch is the filtering operation with the synthesis filter F0 which leads to an output signal which is denoted here as x1 prime prime prime. As we have indicated in the previous slide, the synthesis filters are also assumed to be ideal filters and equal to the analysis filters. In this case F0 is also an ideal brick wall low-pass filter as you see here in the in the first graph. Now if you take the spectrum from the previous slides, the spectrum of x1 prime prime and filter it with an ideal low pass filter F0. Then you end up with the spectrum of the x1 prime prime prime as it is indicated here, that basically only has the red parts and where all the images, the green parts, have been sort of safely removed. So, this is the spectrum of the output signal from the top branch, x1 prime, prime prime. And you see that it basically corresponds to the low frequency portion of the spectrum of the full bands input signal. If you go back 4 or so slides, you see that full spectrum of the input signal.

And the x1 prime prime prime is basically the low pass part of that input spectrum. Notice also that the synthesis filter F0 acts as an interpolation filter following the up-sampling operation. And so, in this case the interpolation filter is indeed a low pass filter but it doesn't have to be a low-pass filter. In the next branch you will see that the interpolation filter indeed will be F1, which is then a band-pass filter.

Slide 21

In a similar fashion, the operations in the second branch of the filter bank scheme can then be traced. The branch with the H1 analysis filter and then the F1 synthesis filter. So, H1 is an ideal band-pass filter, so if you apply this to the input signal, the spectrum of the output of the H1 filter which is noted here as x2. The spectrum of the signal x2 is basically a bandpass filtered part of the spectrum of the original input signal. Notice that this H1 analysis filter precedes the down sampling

operation. So, it also acts as a decimation or anti-aliasing filter. In this case, the anti-aliasing filter is indeed a band-pass filter.

Slide 22

The x2 signal is then down sampled so we have to four-fold down sampling that leads to a signal which is the noted here is x2 prime and the spectrum of x2 prime is basically the spectrum of x2 which has been stretched. So, we have a fourfold stretched a version of spectrum of x2 which is basically the red part and the graph here, plus again the repetitions, the images to make it into a spectrum with a period of 2pi as it has to be. So, this is a spectrum of x2 prime input to the sub band processing again. We assume that the sub and processing doesn't change the signals so it's also the spectrum of the outputs of the sub band processing.

Slide 23

The output of the sub processing is then up samples and so after the four-fold up sampling we have a signal which is denoted here as x2 prime prime and the spectrum of x2 prime prime is basically the spectrum of x2 prime as we had in the previous slide, which is four times compressed. So, we have a four-fold compressed version of the spectrum of x2 prime, where you can see in red original spectrum as the output of H1 and then the green images or repeated versions of that spectrum.

Slide 24

And then finally, in the second branch, we have the filtering with the synthesis filter F1. And so again this is an ideal band-pass filter and the output of this F1 filtering is a signal which is denoted here is x2 prime prime prime. The spectrum of x2 prime prime prime is the spectrum of x2 prime prime as we have in previous slide filtered with this ideal band-pass filter. So, it basically selects a portion of the spectrum of x2 prime prime. And that portion actually corresponds to a portion of the input signal which has been selected by H1. So again, you can see that the spectrum here of x2 prime prime prime is basically the spectrum of the signal at the output of the analysis filter H1. Notice that here the synthesis filter succeeding an up-sampling operation acts as an interpolation filter again. But in this case the interpolation filter is indeed a band-pass filter to reconstruct the sort of band-passed spectrum corresponding to the output of the H1 analysis filter.

Slide 25

From here it's very easy to do the similar analysis for the third branch with H2 and F2, and then for the last branch with H3 and F3. The main observation here is that the spectra of the signals at the output of the synthesis filters are basically equal to the spectra of the outputs of the analysis filters. So, as we have seen it in the previous slide, the output or the spectrum of the output F0 is equal to the spectrum of the output of H0. And similarly as we've also seen it in the previous slide, the spectrum of the output of F1 is equal to the spectrum of the output of H1 etc. And all these spectrums are basically parts of the spectrum of the full band input signal. So, if you add all these signals or spectrum together in the final summation to the far right of the scheme you get an output signal with a spectrum which is indeed exactly equal to the spectrum of the input signal. That property here is referred to as perfect reconstruction. So, the output signal is equal to the input signal. The spectrum of the output signal is equal to the spectrum of the input signal and we've decided that this is indeed the case when we use ideal analysis and synthesis filters. Notice that because of the usage of ideal analysis filters, the down sampling operation doesn't introduce any aliasing effects. The images that you have seen appearing in the spectra do not overlap with the original spectra so there's no aliasing effects and that indeed makes the analysis easy and leads to that simple solution that we do have in the perfect reconstruction where the output signal is equal to the input signal. Obviously when you have non-ideal filters when H0 to H3 are non-ideal analysis filters so that the down sampling operation will introduce aliasing effects. The analysis is much more complicated. It's not as easily drawn as we had in the previous slides.

Slide 27

The previous ten slides looked into ideal filter bank operation and the take home message is that this is actually quite easy because of the ideal filters. There are no aliasing effects due to the down sampling operations and hence you immediately achieve perfect reconstruction. So, the output signal from an ideal analysis synthesis filter bank is equal to the input signal up to at most a delay. Now if you analyze non ideal filter banks where the analysis filters and the synthesis filters are non-ideal filters and hence in the analysis filter bank, you're going to see aliasing effects appearing. Then the question is can you still achieve perfect reconstruction where the output signal is equal to the input signal up to at most a delay and it sounds like this is not going to be possible because of aliasing which is often said to be sort of destroying the information in a signal when you down sample. A signal after a non-ideal decimation filter, some of the information of the in the signal is going to be lost in a sense forever and can never be restored. So, for aliasing we've learned in previous DSP courses that aliasing destroys information which can never be restored. And if you do have aliasing caused by the non-ideal analysis filter bank because of the down sampling, the question is can you still have perfect reconstruction? And the remarkable answer that we'll see in later slides is that this is indeed possible. So, you can have perfect reconstruction filter banks even though you're going to tolerate non-ideal filters in the analysis filter bank and then also in the synthesis filter bank. The basic message will be that the synthesis filter bank has to be designed as a function of the analysis filter bank to remove aliasing effects which are introduced in the analysis filter bank. And that indeed is going to lead to perfect reconstruction filter banks as we're going to study it in the rest of this chapter and the next chapter.

Slide 28

To somehow convince you of the remarkable fact that you can have perfect reconstruction with a non-ideal analysis and then a non-ideal as well synthesis filter bank. Here's a very simple example. And as you will see, it's a simple and not a very interesting example, but in the next slide we will turn it into a more interesting example. Starting point is this, and again it's a very simple scheme with four channels. So, N is equal to 4 and four-fold up and down sampling. So, D is also equal to 4, it's a maximally decimated system in a sense. But the scheme is very simple that it only has delay operations and then up and down sampling and again, delay operations and a summation operation at the end. It somehow looks like an analysis and a synthesis filter bank operation. But now the analysis filters as well as the synthesis filters are pure delay filters, which are not very interesting. They're not frequency selective, so they're not an interesting analysis and synthesis filter bank. Nevertheless, you can easily check that for the simple scheme, the output signal is indeed equal to the input signal up to a delay, which is equal to three. So there's a three-fold delay, basically the three is equal to four channels minus one, hence three delays that you will encounter in the scheme when going from left to right. That's a detail. But remarkably the output is equal to the input up to a delay. And you can easily check that this is indeed the case. If you sort of trace the operations of the top branch of the scheme, the first so called analysis filter H0(z) is equal to 1, so it doesn't change the input signal and then you have a four-fold down sampling. So, the four-fold down sampling basically takes 25% of the input samples and these are transmitted from left to right and then up sampled again where the up-sampling operation introduces zeros in between non zero samples. So, the output from the up-sampling operation and then after an additional delay of z^ (-3). Here you could say for 25% of the original input samples, in between two successive samples you would have three zeros because of the four-fold up sampling. So, you would have a sequence of samples as it is indicated in the graph here, zero, zero, zero, and then first sample u[0], and then again zero, zero, zero, and then the next sample that was kept by the down sampling u[4], etc. Remember that that is basically one fourth of the input samples which are transmitted in the top branch. In the second branch, the H1 so to speak, analysis filter is z^ (-1), so it's a one sampling period delay operation and then again the four-fold down sampling. Because of the one sampling period delay, the four-fold down sampling operation is going to select again 25% of the input samples but then because of the delay another 25% of the input samples. So, if you do the up-sampling again there will be insertion of three zeroes in between two consecutive samples and then a delay operation again with a z^ (-2). The output of the second branch would have u[-1] and then three zeros, and then u[3] and three zeroes, and then u[7], …, u[11], …, u[15], …, on you go. So, it's again 25% of the input samples, but another 25% of the input samples. Similarly, the third branch is going to select another

25% of the input samples, and the last branch is also going to select 25% of the input samples. If you sort of align all these sequences, outputs from the up-sampling operation and delay operations in the synthesis filter bank from top to bottom, the z^ (-3), z^ (-2), z^ (-1) and 1, basically realign these sequences. So that when you add them all up, indeed you can, you can easily check that what you end up with is exactly equal to the input signal u[-3] up to a three-fold delay. So, this is a very simple scheme. Filter bank scheme with analysis filters which are merely delay filters and synthesis filter bank where the synthesis filters are merely the delay filter sites. It's not a very interesting analysis synthesis filter bank scheme. The analysis filters are by no means frequency selective so if you want to use this for adaptive filtering echo cancellation, then the top branch is in no means or in no way going to represent the low frequency part of your problem. So, the analysis filter bank is by no means frequency selective, so it's not a very interesting filter bank, it's sort of the worst filter bank analysis filter bank in terms of frequency selectivity ever on the planet. But still the remarkable thing is that it provides you with perfect reconstruction and in terms of aliasing, of course there's aliasing happening after the down sampling operation. So, if you look into the top branch, you basically take the input signal and do a four-fold down sampling. That will lead to a spectrum where you can observe aliasing effects. And the statement that after aliasing indeed information is lost in the signal, which can never be retrieved, is a true statement because basically after the four-fold down sampling in the top branch, you only have 25% of the information of the samples of the input signal, and the other 75% of the information of input samples were lost. So, the input signal cannot be retrieved or reconstructed from the down sampled signal in the top branch, the sub band signal in the top branch. But as you see it here, the missing information in the top branches is represented in the other branches. So, you can still construct your complete input signal.

Slide 29

Now to make it into a bit of a more interesting a filter bank scheme, let us do an operation as it is indicated in this slide. In between the down-sampling operation and a up-sampling operation, let us introduce a matrix multiplication with a matrix and then another multiplication with the inverse of the same matrix. So we have a matrix F, and F inverse, and we take that four output samples at a particular point in time from the down-sampling operation, which is a vector v, and then this, this v vector gets multiplied first by F inverse and then multiplied by F and so the output of the F or the input to the up-sampling operation, you would have F times F inverse times v, and because of the fact that F times F inverse is an identity matrix, obviously the inputs to the up-sampling operation is still the same vector, the v vector. So any invertible matrix here will do the job. And the job is that basically the output signal is still equal to u at time k-3, so we still have perfect reconstruction in this scheme by inserting any invertible matrix F together with its inverse F. And the matrix that we're actually going to insert is the discrete Fourier transform matrix together with its inverse. And that's a well-known matrix and you will see it represented also in one of the later slides. The important part is we're inserting matrix vector multiplication or two of them with a matrix and its inverse, and the perfect reconstruction property is still preserved.

Slide 30

The final operation which will allow us to interpret what we have here in terms of a true analysis and synthesis filter bank scheme is to reverse the order of the decimation and the expansion or the down-sampling and up-sampling with the multiplications with F inverse and F, as you will see it in the final graph in this slide. So, for instance, in the previous slide you had down-sampling with factor 4, followed by the multiplication with F inverse and then a matrix vector product with a given matrix. So it comes down to multiplications and additions which can be swapped with a down-sampling operation as we have seen it in chapter 2, so basically we're altogether the matrix vector multiplication can be swapped with the down-sampling operation. So you would have the multiplication with F inverse first and then the down-sampling operation, and similarly you would have the up-sampling operation swapped with the multiplication with F, so now the up-sampling operation first and then the multiplication with the F and so now everything that precedes the down-sampling operation. So the delay operations to the far left with the 1 and the z to the -1 and the z to the -2 and the z to -3 and then the multiplication with F inverse that altogether sort of combines into the analysis filter bank and we're going to

analyze the properties of this analysis filter bank in the next slide, and then everything that succeeds the up-sampling operations, the multiplication with the F and then the delay operations and the summation delay operations with z to -3 to one, combines into the synthesis filter bank because of the fact that F will be the DFT matrix. This is referred to as the DFT and inverse DFT filter bank. So the analysis synthesis filter bank as we have it here is the discrete Fourier transform or inverse discrete Fourier transform filter bank, and it is a perfect reconstruction filter bank. So we'll see that it is an analysis synthesis filter bank with some frequency selectivity in the next slide. Not an ideal filter bank so and hence with aliasing effects in the subbands, but nevertheless it provides you with perfect reconstruction. Notice that to the far right of the scheme you still have u[k-3] through the construction as you have seen it in in previous slides. So that tells us that the filter bank is still the perfect reconstruction filter bank.

Slide 31

Let's now look into the analysis filter bank and try to find out what the analysis filters actually look like. In the previous slide, everything to the left of the down-sampling operation belongs to the analysis filter bank. So this is basically where the input signal, and the part repeated here on this slide gets filtered by the delay filters the 1 to z to -3 and then the result of this gets multiplied with the inverse discrete Fourier transform. In this, N is equal to 4 for example. In the general case N is not necessarily equal to 4, we're going to represent this analysis filter bank by means of a vector transfer function which is a vector of which the entries or transfer functions namely the transfer functions, the z transforms of the individual analysis filters. So we have a vector and the first formula here, the vector where the first entry is H0(z), and then second entry is H1(z) and then third entries H2(z) etc.. The meaning of this vector transfer function is basically also that if you take an input signal u[k] time domain and the graph to the left and you filter it by means of the analysis filter bank. Then you obtain a number of output signals and then in the z domain, the z transform of this collection of output signals would be the vector transfer function multiplies to the right with u of z. So if you take the vector transfer function and you multiply it to the right with u of z, you obtain a vector of z transforms of the output signals of the analysis filter bank. Now the analysis filter back consists of the delay filters, and then the multiplication with the inverse discrete Fourier transform. So the input signal first gets multiplied with the delay filters then gets multiplied with the inverse discrete Fourier transform of matrix. So that leads to the first equation. First formula here, where the vector transfer function is basically the inverse discrete Fourier transform matrix, first multiplied by the vector transfer function with the delay filters. So remember if you multiply the vector transfer function with the H to the right with u of z, you can do the same thing to the right of the equation. So you multiply to the right with u of z, then the u of z first gets multiplied with the vector transfer function with the 1, z to the -1, z to the -2 etc. as you have it in the graph to the left. So first input signal goes into the delay filters and then the result gets multiplied with the inverse discrete Fourier transform matrix as you see here in the equation. Now the inverse discrete Fourier transform is a matrix with a special structure. As we have seen it in chapter 2, the review chapter, where every entry in the matrix is a scalar W to a certain power. The W is for an n dimensional DFT or inverse DFT matrix. The W is defined to be e^(-j2pi/N). And then you see in the inverse DFT matrix you see powers of this W in the first row for instance, you would see w to the 0, w to the 0, w to the 0, w to the 0 etc.. And w to the 0 is 1, so in the first row you see all 1's. So every entry in the first row of this matrix is a 1 and then you have the pre-multiplication with 1/N as a sort of normalization or scaling. Similarly, in the second row you would see w to the zero, w to the -1, w to the -2 etc.. Third row you would see w to the zero, to the -2, to the -4, to the -6 to the -8, etc., and on you go. Now, from this equation, then you can specify the individual analysis filters. So the equation basically says H0(z) is equal to, and then you take the first row of the matrix and multiply this first row with the vector transfer function with the delay filters. So multiplying the first row with the vector transfer function. And every w to the 0 is the 1 so that results in the last formula on this slide, that says H0(z) is basically up to the scaling with one over N, it's basically 1 plus 1 times z to the -1 plus z to -2 plus z to the -3 etc. We recognize this as a low pass filter that basically takes an average of the last N samples of an input signal to produce an output signal that is indeed a low pass filter operation. And we will see the magnitude response of this analysis filter H0 analysis filter in the next slide. From here it can also be checked that if you specify H1, by multiplying the second row of

the inverse of DFT matrix with the vector transfer function with delay filters, you would end up with a filter H1(z), which is basically a shifted version of H0(z) shifted in the frequency domain. And every other analysis filter is, again, a shifted version shifted in the frequency domain of H0, as we have it, to a low-pass filter. That is a property that you can try and prove here. You don't have to actually. You can simply accept it here and you see it illustrated in the next slide. Actually we will return to this and in the next chapter and we will derive in a sense this property sort of the other way around from the property derive the fact that it can be represented by means of an inverse discrete Fourier transform matrix. So the details of this will return or we will return to the details in the next chapter. You don't have to worry about the details here.

Slide 32

In this slide, you see the magnitude responses of the individual analysis filters of the analysis filter bank. For the simple example with N is equal to 4 of 4 channel analysis filter bank. Remember that H0 was a low pass filter and basically in this simple example, produces an output signal which is the average of the last 4 input samples, samples of the input signal and that is indeed representative a low pass filtering operation. The magnitude response of such a low pass filter is indicated here and it's basically a sinc function and so you see the magnitude response of H0. And similarly as it was indicated in the previous slide, the magnitude response of H1 will be a shifted version of the magnitude response of H0. So if H0 is a low pass filterl that filters or sort of selects frequencies in a neighborhood of DC of 0 Hertz, then H1 will be a shifted version of that. That is then effectively a band pass filter selecting a different range of frequencies and similarly H2 will be a further shifted version of H1. Then H3 again, a shifted version of H2 and you see that all analysis filters together sort of span the full frequency range from -pi to +pi. So these are the magnitude responses of the analysis filters and the analysis filter bank. You see that these are far from ideal. You can also demonstrate that the synthesis filters which are defined by the F matrix rather than the F inverse matrix, but following a similar derivation you would decide that the synthesis filters have a similar magnitude response. You have it through for the analysis filters. So both the synthesis filters and the analysis filters are far from ideal filters. So if you have the analysis filters followed by a down-sampling decimation operation that will introduce significant aliasing effects. So you're going to see aliasing effects in the subband signals which is bad news. It could hurt the subband processing it being, for instance, and an audio coding and decoding operation will be impacted by the aliasing effects that is bad news, but nevertheless, other than that, the analysis synthesis filter bank is still a perfect reconstruction filter bank. So in spite of the aliasing that is happening in the analysis filter bank, the synthesis filter bank apparently somehow restores the distortion introduced by the analysis filter bank and the analysis synthesis system as a whole indeed has perfect reconstruction, and so the remarkable property or conclusion here is that you can indeed achieve perfect reconstruction even with non-ideal analysis and then synthesis filters.

Slide 33

The last part of the chapter has some of the basics of perfect reconstruction theory and then perfect reconstruction design as we will see it in the next chapter. So, the message from the previous examples is that even with non-ideal analysis and then synthesis filters in analysis and a synthesis filter bank, you can still have a filter bank scheme that has perfect reconstruction. So the question is if there's some magic in terms of matching the synthesis filter bank to the analysis filter bank so that the synthesis filter bank compensates for the aliasing which is introduced in the analysis filter bank. How should you design the synthesis filter bank are sort of matched through the analysis filter bank. So altogether how do you design an analysis filter bank together with a synthesis filter bank so that you have this perfect reconstruction property. We will first look into a very simple case and take a sort of straightforward approach. So 2 channel case with 2-fold down-sampling. So maximum decimated case. Very simple example where will observe that even for this very simple example, the straightforward approach leads to sort of quite complicated design exercise which is not easily extended or generalized then to the N channel case. And so for the general case especially we take a different approach which will be based on polyphase decompositions which then leads to a very simple perfect reconstruction design criterion that we'll use in the next chapter to actually design perfect reconstructing analysis/synthesis, filter banks.

Slide 34

So here's a very simple two channel case, for example, to begin with. So you have an analysis filter bank with 2 analysis filters, H0 and H1, and then a maximal decimation, two fold for the two channel case decimation. At the analysis side and at the synthesis side, you have two-fold up-sampling and then synthesis filters F0 and F1, again, notice that there's no subband processing. Remarkably, we were designing the filter bank to do subband processing, but we're assuming that the applications, as we have seen some of them were such that the subband processing is basically not there can be or you can act as if there's no subband processing, and then, you will have to analyze perfect reconstruction. So we have a simple two channel filter bank, two channel up and down sampling. And the question is what is the output signal? Can the output signal be a delayed version of the input signal to give us perfect reconstruction? This is what we have to investigate. Now the output signal, the y[k] and the time domain can be specified by means of Z transform Y(z) in the first formula on the slide here. This is where you have to use your formulas from chapter 2 to see what an up and down sampling operation basically does, to Z transform of an input signal. And by using these formulas you can easily derive that Y(z), the z transform of the output signal, is equal to, and then you have the equation with two terms and the first term has an expression with H0, F0 and H1 and F1, which is denoted here as T(z) times U(z), so this is like a linear transfer function to U(z) gets multiplied by a transfer function T(z). And then you have a second term which again has an expression with H0, F0, H1, F1, which is denoted here as A(z), and that multiplies U(-z), which is sort of remarkable. It's something that we haven't come across too often probably. And it's important actually if we start with the second term to see what the meaning is of this U(-z), remember that if you have a Z transform and you evaluate it in the unit circle, so you replace z by eˆj, then you see the spectrum of the signal. And so if we do that here we evaluate U(-z) under unit circle. You replace z by eˆj. This is the footnote at the bottom of the slide. So, in U(-z), you evaluate for z is equal to eˆj. What you end up with is U of e to the j+pi. And the meaning of that is basically that the spectrum of the input signal has been shifted over an amount pi. So that means that for instance, the DC signal is shifted into the Nyquist frequency or more generally, low frequency content is shifted into high frequency content and vice versa, high frequencies are shifted into low frequencies. This is basically aliasing at work, aliasing is where you introduce frequency content at places where this frequency content is not represented in your input signal. So, and because of this A(z) is indeed referred to as the alias transfer function. So it's transfer function applied to the shifted version of the input spectrum which is represented here by the U(-z). So A(z), remember is the alias transfer function it multiplies U(-z) and then we go back to the first term. First term has a transfer function T(z) that multiplies the input Z transform U(z), so this is a usual sort of linear filtering operation. The T(z) is somewhat remarkably referred to as the distortion function but this is in a sense of linear distortion as you see it or you can interpret the T(z) as a linear filtering operation. So, distortion is basically amplitude and phase distortion, linear amplitude and phase distortion, and so T(z) is the distortion function and remarkably you can also identify the T(z), if you look into the details, T(z) is equal to H0 times F0 plus H1 times F1. You go back to the initial scheme that is basically the transfer function. If you were to remove the up and down sampling operations from the scheme then you would see H0 timesF0 in the top branch. And H1 times F1 in the lower branch and then you add these two terms and that is indeed the expression for T(z). So we have an alias transfer function, A(z) and a distortion function T(z), which somewhat remarkably is equal to the transfer function with the or for the full scheme with the down-sampling and up-sampling operations removed.

Slide 35

From the expression on the previous slide, you can specify the so-called perfect construction conditions for the analysis and the synthesis filter back together. So if in the formula on the previous slide, basically the output signal has to be equal to the input signal up to a delay, then first the alias transfer function A(z) obviously has to be equal to zero. So that the filter bank operation is as what we would say alias free. So A(z) has to be equal to zero and that gives you sort of a condition or a constraint on the analysis filters H1, H0 together with the synthesis filters F0, F1. When the alias transfer function is equal to zero, then what remains is basically Y(z) is equal to the distortion function T(z) times U(z) as you had in

the previous slide. So this is where the complete filter bank, even though you have up and down sampling operation, starts behaving as a linear-time-invariant system with a transfer function T(z). Now if you on top of that insist that the T(z) is a pure delay so that the output signal is a delayed version of the input signal, then T(z) indeed has to be equal to z to the -, for some as you see at the bottom of the slide. And then so this condition together with a condition for alias-free operation A(z) equal to zero, that would be sort of the overall constraints or conditions for perfect reconstruction. And so then the question is what are analysis filters, 0, H1 together with synthesis filters F0, F1 that satisfy these two equations so that the alias transfer function is equal to zero and the distortion function is equal to a pure delay.

Slide 36

A solution to the perfect reconstruction design equations from the previous slides, A(z) is equal to zero and T(z) is a delay, or A(z) and T(z) are functions of the analysis and synthesis filters. So solution to that set of equations is provided in this slide and the message is not so much to remember this solution only to give an example of a solution, which in itself already looks pretty complicated for a very simple only two channel design problem. And actually it also took quite a while to come up with this solution only in the 80s the solution as you have it here was published. So it is given here as an example and there's no true derivation or anything. But if we briefly go over it, it says the synthesis filters the F0, F1 can be derived from the analysis filter. So if you first design the analysis filters H0, H1, then F0 is derived from H1, it's actually equal to H1(-z) and remember the meaning of this -z is basically that you shift that frequency response or spectrum over pi. So if this is where you see if it is a high pass filter then it's it gets shifted into a low pass filtering and that is your F0 first low pass synthesis filter. And similarly, for F1 which is derived from H0, H0 being a low pass filter. And then again this is shifted into a high pass filter F1. If the synthesis filters are selected like this then you can quickly prove that the alias transfer function is equal to zero. And the remaining question is how do you design the H0 and H1. And the rest of the statement here is that you have to pick an H0 which satisfies a particular property which is referred to as H0 being power symmetric satisfies an equation as you have it here in the middle of the slide, ignore the details truly. If you have an H0 which is derived or which satisfies this equation, then you can derive the corresponding H1 from there. Basically, you start from the impulse response sequence of h0, and then you reverse the frequency response, the impulse response, and modulate it with a sequence plus one minus one, plus one minus one plus one minus one, that's the formula in the third part of the slide here. So you derive the impulse response sequence of the h1 analysis filter from the impulse response of the h0 analysis filter. And again, because of the modulation, basically a low pass H0 filter is turned into a high pass H1 filter. If you design the H0 to be power symmetric and you make H1 such that it satisfies this third equation, then you can again prove that the distortion function T(z) is equal to 1. An additional property of the solution here is that the analysis filter bank, the H0, H1 satisfy an equation, the last equation on the slide here that basically says that the squared magnitude response of H0 plus the squared magnitude response of H1 is equal to 1, and we've seen an equation, something like this in chapter5. And we have set of filters that satisfies this equation is said to be power complementary or represents a lossless system. So here H1 and H0 represents a lossless system. We will return to this type of analysis and then also synthesis systems and these will be referred to as paraunitary filter banks analysis and synthesis filter banks. You will return to this in the next chapter. The message here is not so much to look into the details of the solution here, only to sort of appreciate that the solution is already pretty complicated and we're only talking a two-channel case two-fold down an up sampled two-channel analysis and synthesis filter bank.

Slide 37

Question now is whether you can generalize the design procedure for the two-channel case to perhaps first the three channel case and the four channel case, but then in general the N channel case and we're still looking for simplicity in a sense to the maximally decimated case where the up and the down sampling factor the D is still equal to N the number of channels. So in the maximally decimated case the question is what are the perfect reconstruction design equations. And to get there first of course you have to derive an input output equations similar to the input output equation for the

two channel case as we added a few slides back. And so in this case you do a similar derivation and the outcome of the derivation is provided here is the first formula Y(z), the z transform of the output signal is again a sum of basically two terms. First term is at T(z) times U(z), where U(z) is the z transform of the input signal and the T(z) is again a distortion function and the distortion function then plays a role as a linear-time-invariant system that filters the input signal U(z), the T(z), again, you can check it in the formula corresponds to removing the up and down sampling operations in the scheme, the filter bank scheme, which is given here in the slide also if you remove the up and down sampling operations to down sampling by four in this example and up sampling by four what remains is a transfer function H0 times F0 in the first branch plus H1 times F1 in the second branch etc.. This is basically up to a scaling the distortion function and then in the perfect reconstruction condition you would have that this T(z) distortion function should be equal to a pure delay. Then you have the second term which is again an aliasing term. The difference now is that this term in itself is a summation over N-1 terms. In the two-channel case you had one alias component which basically had the input signal spectrum which was shifted over an amount pi or pi is basically 2pi divided by 2 where 2 is the number of channels. Here you're going to have different shifted versions of the spectrum of the input signal all shifted over multiples of basically 2pi divided by N, now where n is the number of channels you have in your filter bank system. So you would shift the input spectrum over 2pi divided by N and then over 2 times 2pi divided by N and then 3 times etc. And this is basically the meaning of this summation term. You have the U(z) times W to the power n where the W was defined in the previous slide where we had the inverse discrete Fourier transform matrix and this basically represents the input spectrum shifted over a certain amount and this shifted input spectrum basically or input signal gets multiplied by an alias transfer function. Alias transfer function, which is denoted here as An(z) for the nth term in the summation. So, the observation now is we have a distortion function and then we have basically N-1 alias transfer functions. And in the design equations the T(z) distortion function should be equal to a pure delay and then all N-1 alias transfer functions should be equal to zero. That all together is basically N equations and all of your analysis filters the H(z) and all of your synthesis filters the F(z) and it goes without saying that this is a quite complex set of equations which is difficult to solve. In the two-channel case it was already difficult to solve. In the N channel case, it's almost impossible to solve it from here. So the conclusion is we need a different approach to sort of construct or perfectly construction design equations and that will be given in the next few slides.

Slide 38

The message from the previous slide is that we need a different representation to represent the analysis filter bank and the synthesis filter back to be able to derive sort of an easier set of design or perfect reconstruction conditions or design equations and this is provided this alternative representation is provided in this slide, and it's based on polyphase decompositions of all of the analysis filters, as well as all of the synthesis filters, and this slide we're still looking into the maximally decimated case. So we have in general an N channel filter bank where the up and down sampling factor is equal to N so the D for up and down sampling is equal to the N maximally decimated case. Now the starting point for this is basically the first equation in this slide, where to the left you see the vector transfer function with all of the analysis filters. So first entry is H0(z) and then the second entry will be H1(z) and the last entry will be the last analysis filter H_N-1(z) and now for each of these transfer functions we're going to substitute a polyphase decomposition, an n-fold polyphase decomposition, in this maximally decimated case, where D=N, so it could be a default polyphase decomposition with D=N, so we're using an n-fold polyphase decomposition first for H0, and in the equation the H0 is basically the first row of the matrix which is denoted here is the bold E matrix. First row of this matrix times the vector to the right of the equation and in this vector you have the delay operation. So if you multiply the first row in this bold E of z matrix, E of z to the N, a matrix where each entry is indeed a function of z to the N because we're doing an n-fold polyphase decomposition. If you multiply this first row of this matrix with the delay filters 1 and z to the -1 and z to -2 etc., you have the definition of the n-fold polyphase decomposition of the H0 filter. If you do not remember polyphase decompositions etc., go back to chapter 2 for the review. Similarly, you have in the second row of this bold E matrix you're going to have the polyphase

components of the second analysis filter H1(z) will be equal to and then the entry is in the second row of this bold E matrix again times the delay operation. So first component times one plus second component times z to the minus one etc. So this is the starting point here. Now also remember that this vector transfer function with the H which we have also come across in the previous slide. If you post-multiply it with U(z) that basically represents the filtering operation of an input signal U with all of the analysis filters. Now if you multiply the left-hand side of the equation by U(z), you also multiply the right-hand side of the equation by U(z), multiply it to the right and then what you basically have is when you read it from right to left you would have the U(z), which first gets filtered by the vector transfer function with the delays. So with the 1 and z to -1 and z to -2 etc. And then you have the multiplication with the bold E matrix, and this is what you see in the scheme at the top of the slide. But now you read the scheme from left to right instead of from right to left. So you have the input signal to the far left u[k] in time domain. This signal gets filtered by the vector transfer functions with the delay filters. So it gets filtered by a 1, by z to the -1 filter, by a z to the -2 filter, by a z to the -3 filter in this example. And then the result of that gets multiplied by the bold E of z to the 4, in this simple example. Then you have the 4-fold down sampling and that is representative of the analysis filter bank. So the analysis filter bank has basically just been replaced by a different but equivalent representation with this bold E matrix which has all the polyphase components of the analysis filters. Similarly for the synthesis filter bank, we're going to replace the synthesis filters by a polyphase representation, instead of a bold E matrix, we're going to have a bold R matrix and instead of having the polyphase components of the synthesis filters in the rows of the bold R matrix, you're now going to see that they have to be stacked into the columns of the bold R matrix. So the last formula on the slide here is basically a row vector with F0 F1 F3 etc. is equal to a row vector of delay operations. And these delay operations multiply components in successive columns of the bold R matrix and that again corresponds to, in the scheme at the top of the slide, taking the output of the up sampling operation multiplying first with the bold R matrix which is a function of z to the N, so z to the 4 in this case, and then filtering with delay filters and summing the results to produce the output signal. So you have to look into the details of the formulas here and convince yourself that these formulas indeed corresponds to the block scheme as you have it here in the slide. This is extremely important. It's non-trivial perhaps but you have to try hard to understand this properly. So look at the block scheme compared to the formulas and make sure convince yourself that you see that this is indeed exactly the same operation and the message here is that is that basically we haven't done anything so far. We're still where we still have to design an analysis and a synthesis filter bank. Only we've used a different representation of the analysis and synthesis filter bank. The analysis filter bank is now represented by the bold E matrix, where you have polyphase components and the rows for successive analysis filters and the synthesis filtered bank is represented by the bold R matrix, where you have polyphase components of successive synthesis filters in the columns of the bold R matrix.

10 Chapter 13

Chapter 13

10.1 Slide 9

In general, the motivation for using polyphase decomposition is that by using a polyphase decomposition you split a general filter into polyphase components which can each individually be typically swapped with an up or down sampling operation. As we have seen it in chapter 2, and this is referred to as the noble identities and in chapter 2 and these novel identities will be applied here for the very simple 4-channel case as you also had it in the previous slide. In the previous slide you would have the filtering with the bold E first and then the down sampling operation four-fold down sampling operation. The filtering with the E is a filtering with E of z to the 4 in the previous slide followed by a four-fold down sampling operation. This is where the noble identities tell you that you can swap these operations. You can do the

four-fold down sampling first as you have it in this slide and then this is followed by the filtering and now the filtering is not with E of z to the 4 but of the filtering with E of z to the power 1 basically. Similarly in the synthesis part in the previous slide we would have four-fold up sampling followed by a filtering with R(z) to the four. Noble identities tell you that you can swap these two operations you can have the filtering first and then this will be a filtering with R(z) to the 1 instead of R(zˆ4) and then this is followed by the four-fold up sampling. So this is the resulting scheme. Here you have the input signal which is filtered by delay filters down sampled. Then you have the E and R and then up sampling delay filter summation. And the question now is with this alternative representation of an analysis synthesis filter bank or again E(z) is representative of the analysis filter bank and R(z) is representative of the synthesis filter bank. Can you derive simple equations that provide you with a system where all aliasing effects are cancelled, which is alias free? And on top of that where you would have a distortion function which is equal to a pure delay so that you have perfect reconstruction, now this is where you can already sort of predict that the current representation is going to lead to something which could be quite simple if you go back to slide 24. Then in slide 24 you basically had a system which is similar to what you have here, only in fact, without the E(z) and the R(z), you would have the input signal, which gets filtered by the delay filters 4-fold down sampling. And then let's say no, E and R 4-fold up sampling and delay filtering and summation. And we, we've demonstrated in slide 24 that this actually responds to a perfect reconstruction system where the output signal is indeed u[k-3] in this four-channel case at a three sampling period delayed version of the input signal. So that would correspond to the perfect reconstruction system. So if the E and R were not there we would have perfect reconstruction. So with the E and the R in place you could sort of immediately predict that if the E and the R are each other's inverse, then E times R o R times E is going to be an identity matrix and then basically is removed from the scheme. And then indeed you have perfect reconstruction. So somehow from here you can already predict that if the R matrix is the inverse of the E matrix then you would have perfect reconstruction. And in the next few slides we're going to look into the details, the actual requirements for perfect reconstruction, which will lead to a very similar, slightly more general conclusion actually.

10.2 Slide 10

Let's first look into alias-free operation. So the question is when is the filter bank and alias-free filter bank if you go back to slide 33 where you have the expression with alias transfer functions N-1 in general of them. The question is when are all these alias transfer functions exactly equal to zero? And the statement here is in terms of the polyphase decomposition matrix is the bold E and R matrix. The statement here is that the necessary and sufficient condition for alias-free operation. To have an alias-free filter bank is that the product of the bold R(z) matrix times the E(z) matrix is a pseudo-circulant matrix. This is a necessary and sufficient condition; we're only going to prove in the next slide. We're sort of informally prove the sufficient part of this statement, not the necessary part of the statement. So, but the statement is actually quite simple. It says R times E is a pseudo-circulant matrix. Question then is what is a pseudo-circulant matrix. We've seen in a previous chapter what a circulant matrix is, where each row is basically the previous row where every entry is shifted one step to the right, and then the entry that sort of is removed from the matrix to the right-hand side is circulated back into the left-hand side. That is a circulant matrix. Then a pseudo-circulant matrix in this case here is a circular matrix where on top of anything else, all of the elements/entries below the main diagonal are multiplied by z to the -1, so you see that in this pseudo-circulant matrix as you have it here, you have a first row p0(z) p1(z) p2(z) p3(z), notice that these are functions of z, as R and E are also functions of z, so the first row is p0 p1 p2 p3 and then the second row you have the p0 which gets shifted one step to the right. So it appears in the two-two element, second element in the second row, the p1 is shifted into the two-three elements, third element in the second row, and then the p2 is shifted in the fourth element. In the second row, the p3 in the first row, it's shifted to the right, it's removed from the matrix but then it's circulated back in, so that left. So the first element in the second row is p3 again, but this is where it gets multiplied by z to the -1 because it's an entry below the main diagonal. On you go you can check the other entries matrix and so this is in the end the definition of a pseudo-circulant matrix still for a very simple four-channel case. So, the statement here is a necessary and sufficient

condition for alias-free operation is that R times E is a pseudo-circulant matrix. Notice that we have in this condition R times E so you multiply E to the left with R matrix and we always read formulas in a sense from right to left where you start from for instance U(z) which gets filtered by something and then something else, if you go from right to left in an equation, whereas in a scheme as you have it at the top of the slide you would see the operations from left to right. So there this is where you see the E(z) appearing first and then the R(z), so a vector signal gets multiplied by E(z) and then by R(z) in the scheme in the formula which multiply R times E times the input vector, where the vector first gets multiplied by E and then by R, so do not be confused. In the scheme you would see the E to the left and the R to the right and the formula here the condition for alias-free operation, you see the R to the left and the E to the right. So for necessary and sufficient condition for alias-free operation is that R times E is a pseudo-circulant and matrix. If the filter bank is alias-free, then everything that remains is a distortion function, T(z), and then it turns out that this pseudo-circulant matrix also defines the distortion function in that the polyphase components of this distortion function are found in the first row of this pseudo-circulant matrix at the p(z), the p0, p1, p2 and p3 in this four-channel case for example, are indeed the polyphase components of the resulting or remaining distortion function. And so the sufficient part of this statement is sort of informally proved in the next slide to make it a bit more acceptable.

10.3 Slide 11

An informal proof of the sufficient part of the statement in the previous slides, as provided to in this slide and this is actually very simple in the previous slide you had in between the down sampling and up sampling operation, you had the filtering with the poly phase E and then the filtering with the poly phase R altogether in the formula, whereas in the block scheme you would see the E to the left and the R to the right, and the formula. This would be a multiplication with R * E that R to the left and then the E to the right. So again do not be confused here.

This here this is a matrix transfer function of Z to the one mobile identities again tell you that this can be swapped with the downsampling operation, so you would have to filtering with the matrix transfer function first, but now the Z is replaced by Z to the four, so you would have a filtering by polyphase R of Z to the four times polyphase E times Z to the four, and then the downsampling operation, and then still the upsampling operation to delay filters on the summation. Etcetera.

Now the question is the remaining question is then whether you can still swap the delay filters at the beginning of the scheme or to the left of the scheme with the filtering with RZ to the 4 * E of Z to the four. And actually this can also be done so you can also swap these two operations in a sense if you look into the formula in the middle of the slide. So you would have the sort of first left hand side of the of the scheme corresponds to U of Z. As you see it in the left hand side or the OR the left hand part of the equation U of Z gets first filtered by the vector transfer function with the delay filters to one Z to the -1 Z to the -2 Z to -3 and then gets filtered by RFC to the 4 * E of Z to the four. Now if for r * E You substitute the pseudo circulant matrix as you have it in the previous slide, you can easily check that that indeed corresponds to the delay vector transfer function multiplied by always the same expression which is indicated here as to T of Z, which is immediately in a polyphase decomposition form times the U of Z and this is merely a matter of substituting the pseudo circulant matrix from the previous slide. So you have to do this for yourself. But the conclusion is the interpretation of the left hand or I should say right hand part of the equation here is that U of Z first gets filtered by T of Z of which the polyphase components are P OF 0, P1, P2, P3, then gets filtered by the effect your transfer function with the delay filters.

And then you have the rest of the operations in the original scheme. The downsampling of sampling and delay filters and summation. So altogether that leads to the scheme as you have at your at the bottom of the slide, you take your input signal, U of K it first gets filtered by T of Z filter.

Specify it in terms of its polyphase decomposition and the formula in the middle of the slide. Then that goes into the delay filters 1, Z to - 1, Z to - 2, Z to -3. Then you have to down sampling for fall down sampling for fold up sampling delay,

filter summation.

And what you see now appearing is basically that the input signal gets filtered by T of Z and then everything else is basically what we had in the previous slide, then which we have identified as a perfectly reconstructing a system where so everything from the initial delay filters down sampling, up sampling and delay filter submission is basically takes a signal and produces adds it up the same signal up to 3 sampling period to delay. So at the outputs overall you're going to see the input signal filtered by T of Z and then subject to a three sampling period delays.

So that is a linear filtering operation with a distortion function T of Z times, basically Z to the -3, so the whole system starts acting as a linear time invariant system and is indeed free of aliasing effects. So you have an alias free filter bank that acts as a linear filtering system with the distortion function T of Z and again the polyphase components of the T of Z are specified as the entries in the first row of the pseudo circulant matrix.

10.4 Slide 12

The remaining question now is once you have alias free operation when you have perfect reconstruction. So if you have the alias free operation, you have the resulting distortion function: the T of Z. And so now the question is when is the T of Z pure delay so that we indeed have perfect reconstruction.

Now from the previous slides, you can immediately state that unnecessary and sufficient condition for perfect reconstruction is that the R times E matrix this pseudo circular matrix from the previous slide that now takes special form as it is indicated here in the formula in the middle of the slide R times E of Z is the pseudo circulant matrix, which is now a sparse matrix with 0 entries and then two parts which are basically an identity matrix scaled by Z to the minus delta or multiplied by Z to minus delta and then another identity matrix multiplied by Z to minus delta -1.

The reason for this is very simple. In the previous slides, we've stated that in the pseudo circulant matrix the entries in the first row corresponds to the polyphase components of the distortion function. Now if the distortion function is to be a pure delay and a pure delay has an impulse response 0000 and then along somewhere and then again 0000. And knowing that the polyphase components are basically downsampled versions of this impulse response.

You can easily decide that the polyphase components of this if T of Z is a pure delay, all of the polyphase components are going to be exactly 0 and have only zeros in their impulse response, except for one polyphase components, which is going to have all zeros and then 11, and the in its sample response.

In itself, then it's going to correspond to a pure delay, say Z to the minus delta. So if the T of Z has a polyphase decomposition where all of the polyphase components are zero except for one of the polyphase components, say PR of Z is a pure delay, say Z, to the minus delta. Then you can pull the first row of the polyphase matrix, the pseudo circle, and that matrix I should say in the previous slides with the zeros and then the Z to the minus delta and the R or the R + 1. I should say position and then construct the rest of that matrix based on the definition of a pseudo circular matrix where the Z to the minus delta -1 is basically the Z to the minus delta and then an additional multiplication by Z to the -1 for those elements below the main diagonal of the matrix.

So from the previous slides you can immediately decide that the necessary and sufficient condition for alias free operation is what you have is the as the formula in the middle of the slide here the observation is that this is basically sort of an identity matrix where some of the columns are have been permuted and some of the rows have been permuted accordingly. Sort of a permuted version of an identity matrix, and indeed a special cases where R is equal to 0 and then R times E is indeed an identity matrix multiply by Z to the minus delta and the conclusion will actually be that by sort of doing this permutation of rows and columns to have the more general form as you have it in the middle of the slides doesn't actually buy you much in terms of design freedom and so it's perfectly OK to stick to the simplest form as you have

it here at the bottom of the slide. That basically says R times E is equal to an identity matrix up to a delay, and so if the delay we're not there then you would say R is has to be the inverse E and that indeed is the results as we have predicted it actually in slide 35. So if you go back to slide 35 already there, we sort of predicted that if R&E are each other's inverse, then then indeed you would have perfectly extraction so but now in a more general statement, we say R times Z doesn't have to be an identity matrix, it can be an identity matrix up to a delay, and then the identity matrix can also be permuted into something as you have it in the middle of the slides. But we're not actually going to exploit that sort of generality. So in later slides and in the next chapter, the perfect reconstruction condition that we're going to be using is this remarkably simple condition. R Times E is equal to an identity matrix up to a delay.

10.5 Slide 13

In the previous slide, we derived the perfect reconstruction condition for maximally decimated filter banks over the capital D up and down sampling factor is equal to the capital N (number of channels) in the filter bank, it turns out that this is very easily generalized for oversampled filter banks for the down sampling factory is smaller than the number of channels and in the example in the scheme here in the slide you would have 6 channel filter bank with only four fold instead of 6 fold up and down sampling it turns out that you can do a very similar derivation. The only difference is basically that in terms of the polyphase decompositions that you're going to use to define the polyphase E matrix and the polyphase R matrix. We're going to use default polyphase representations.

So in the first formula on the slide here, you're going to have the vector transfer function H of 0 of Z, and then H1 of Z etcetera. And in the first row of the polyphase E matrix you're going to see D polyphase components of H of 0.

So in each row of the E matrix, you're going to see a default polyphase decomposition of one of the filters of the analysis filter bank. So the number of columns of the E matrix is the capital D, whereas the number of rows is the number of channels filters you have in your analysis filter bank is the capital N so the dimension of E is N * D it has N rows and D columns. It has more rows than columns and so this is referred to as a tall thin matrix.

And similarly, R matrix everything is going to be reversed. Again, we're going to use T fold ball phase decompositions, but in this case the polyphase decompositions for an analysis filter are going to sit in a column of the R matrix, so you're going to have N columns, and in each column you're going to have capital D entries.

So the R matrix is going to be D times N matrix with more columns and rows and have this is referred to as a short fat matrix.

10.6 Slide 14

A similar derivation then leads to a perfect reconstruction condition where the simplified form similar to what we used in slide 13 for the case R is equal to 0 simplified form, is then basically the bold space R matrix times the bold face matrix is again an identity matrix, but now if you check dimensions of the matrices. This is going to be at times D identity matrix and then again multiply it by the Z to the minus delta.

So again, a beautifully simple and a sense perfect reconstruction condition. The only difference is basically in the details and the dimensions of the matrices involved. So again, R is short fat matrix D times. N, E is a tall thin matrix N times D and then the product of those is D times T matrix.

And the perfect reconstruction condition says this has to be an identity matrix d * T identity matrix.

So again, the result is beautifully simple. We're going to use this result in the next chapter to do actual analysis, synthesis, filter bank design. We're going to consider the maximum decimated case where D is equal to N, and we'll see that because

of the fact that we're multiplying square matrices together and then say the product of these two square matrices has to be an identity matrix that this this basically will involve of matrix inversion, which is sometimes problematic in terms of matrix transfer functions and doesn't offer a lot of design flexibility. Actually, there's in the Oversampled case because of the fact that we're multiplying the short fat matrix together with a tall thin matrix that sort of avoids or does something which is more general the matrix inversion and you will see that that indeed leads to more design flexibility, so that will be discussed in the next chapter where we will look into analysis, synthesis, synthesis, filter bank through design procedures.

10.7 Slide 16

An important aspect is that when we're designing perfect reconstruction filter banks, we're actually combining 2 design targets. On the one hand, we're designing filters in the sense of Chapter 4, where we'd like to have frequency selective filters with, for instance, a large stop bands attenuation a small passband triple a steep transition band, etcetera. This is important in view of the application that we're actually doing. Think of an audio coding application for instance again where each sub band signal is encoded.

And this encoding can suffer from aliasing effects when we have the downsampling after non frequency specific filters for instance. So in terms of the application it is important to have especially a good frequency selective analysis filter bank at the same time we would like to have a perfect reconstruction filter bank so that the synthesis filter bank is matched to the analysis filter bank to give us perfect reconstruction.

Addressing these two design targets at once, is truly the challenge. Designing filters frequency selective filters as such is not a difficult task. We've seen that in Chapter 4 and designing perfect reconstruction filter banks is also as such not a difficult task we've seen.

Simple examples in the previous chapter.

First, the delay filters and the TFT filter bank, which are not great filter banks, but nevertheless they offer perfect reconstruction, so perfect reconstruction as such is not too difficult and designing frequency selective filters is not too difficult. But combining the two that is truly the challenge which will be addressed in this chapter.

Note that we're designing perfect reconstruction filters here. Another approach is to do near perfect reconstruction filter bank design, where basically the filter specifications are optimized together with sort of a minimization you could say of aliasing and distortion and the two aspects then captured and one on the same optimization problem, and then solved. But this is a design procedure which will not be covered here or addressed here in this chapter.

10.8 Slide 18

Let's get started with this design problem that has first look into the maximum decimation case. So maximum decimated filter banks the D downsampling factor is equal to the capital N the number of channels. In this case the perfect reconstruction condition is r * E is equal to an identity matrix up to a delay where the identity matrix is given N by N Matrix and both R&E are also N by N of square matrices.

A straightforward design procedure would then read as follows. In a first step, you design, for instance, all of the analysis filters 1 by 1. So you first design the first low pass filter, then the band pass filter, etcetera. On you go until you have designed each and every filtering, the analysis filter bank so that is basically based on what we have seen in Chapter 4.

Then once you have all the analysis, filter individual transfer functions. Then for each analysis filter transfer function you can do a polyphase decomposition and then each such polyphase decomposition basically defines one row of the

bold face E of Z matrix. The polyphase matrix representing the analysis filter bank. So then we have in each row E of Z basically an analysis filter and polyphase decomposition so that determines the full E of Z matrix and then finally, in order to satisfy the perfect reconstruction condition, you could compute the synthesis filters represented by the RFC by basically inverting the E of Z matrix and then perhaps add a delay to make things causal, as you will see in later example but so basically everything you have to do here to design the synthesis filters is inverts to E of Z matrix and then the columns of the R of Z matrix basically provide you with the polyphase components of the synthesis filters. The individual synthesis filters so that fully specifies the synthesis filter bank.

Now, we're going to focus on FIR filters in the analysis filter bank because for FIR filters, we can easily do polyphase decompositions, for instance.

Nevertheless, with FIR filters in the EZ inverting the EZ matrix, assuming this inverse indeed can be computed, will in general still lead to an IIR matrix transfer function, R of Z Infinite impulse response matrix transfer function R of Z and at the same time stability could be concerned here when we're inverting this matrix transfer function E of Z further details on this will be given in the next slide.

10.9 Slide 19

The previous slide has a matrix transfer function. Inversion something we may not be all that familiar with. Here's a very simple example to start with. Consider a simple case where the matrix is a one by one matrix. So just transfer function. So an FIR finite impulse response transfer function example is given here. E of Z is 2 - Z to the -1. So this is an FIR transfer function. If you invert it, that would be R of Z the result is 1 / 2 - Z to the -1, and so this is an infinite impulse response transfer function rational transfer function rather than an FIR polynomial transfer function. So except for special cases and FIR transfer function, inverse will always lead to an infinite impulse response transfer function.

For an infinite impulse response transfer function, we should at the same time be worried about stability, meaning that in the original FIR transfer function you can have zeros that can safely lie outside the unit circle, but if you invert the transfer function then these zeros become poles in the IIR transfer function and then poles have to be stable ends have to lie inside the unit circle. So, this is something to be worried about. Stability is something to be worried about.

Let's now move on to the N-by-N case and but then again a simple example. First with a two-by-two FIR matrix transfer function. So, the example that you have here you E of Z is a two by two matrix. Each entry is of this two by two matrix is an FIR transfer function. So we call E of Z and matrix FIR transfer function. So this is a matrix which has one parameter which is equal to the Z. But still you can't compute the matrix inverse even when it has parameter Z and the result is provided here R of Z is equal to and then you see the matrix if you could multiply the R of Z and the E of Z together to convince yourself that the results is indeed an identity matrix.

Remarkable thing here is that the inverse of the FIR matrix transfer function is again an FIR matrix transfer function. So every entry in the R of Z matrix is indeed an FIR function and the reason for this is basically that the determinant of the original two by two transfer function E of Z is equal to 1.

If you have your E of Z with determinants equal to 1, the inverse would be sort of a reordering of the entries in the E of Z matrix, and then you divide by the determinant. But then the fact that the determinant is equal to 1 basically leads to the conclusion that the inverse sort of R of Z matrix is indeed still an FIR matrix transfer function, so the conclusion is in general, when you invert an FIR transfer function and certainly so for instance a one by one matrix transfer function you would end up with an IIR transfer function where you are concerned or should be concerned about stability, but in the general N by N case with N larger than one there are certain matrices that have FIR Matrices. I should say that have an inverse which is again an FIR matrix. And this is remarkable.

To understand this better, you can also compare this to the inversion of integers and integer matrices. So if you take one integer for instance, the example here is E is equal to two inverted, the inverse is 1 / 2 is not an integer number, but if you consider matrices again a two by two matrix you can construct a specific matrix or pick a matrix from a certain set of matrices where the inverse it again an integer matrix and there's a small example given here two by two matrix with entries 564 and five. If you compute the determinant, that's the five by 5 or 5 * 5 , - 6 * 4 determinant is again equal to one, and then if you compute the inverse, the inverse indeed it has again only integer numbers.

Sldie 20

Returning to the design procedure, two slides back where we first designed the analysis filter bank. So basically the E of Z matrix transfer function and then drive the synthesis filter bank from there by inverting the E of Z matrix to give the R of Z matrix. The question could now be can we design or built an E of Z matrix, which is an FIR matrix transfer function N by N in the maximally decimated case here such that the inverse of the FIR E of Z is again an FIR matrix transfer function enhance guaranteed to be stable because FIR transfer functions. You can never have on stable poles, so then the synthesis filter bank would indeed be guaranteed to be stable. So the question is can we do this? Can we find such an FIR EZ? Or how can we find such an FIR EZ ?

Now it turns out that this is indeed something that can be done if you look into that figure here to the top right of the slide basically we're considering EZ and the set of all possible matrix transfer functions E of Z in this world, and then a subset of this set of E of Z's which is the set of finite impulse response matrix transfer functions E of Z. And then again a subset of this subset which is referred to as the setup of unimodular FIR EZ matrix transfer functions and unimodular E of Z matrix transfer function is defined basically as a matrix transfer function, of which the determinant is a constant times a delay Z to the power D.

And it's basically because this fact and you've seen an example of this in the previous slide where the determinant was indeed a constant equal to 1 when in the more general case the determinant is a constant and then also times Z to the power D if you are to invert a matrix the determinant is inverted but a constant inverted is still a constant, and then Z to the D inverted. This is Z to the minus D and this is basically still FIR. So we're defining a set of FIR matrix transfer functions which refer to as unimodular and the definition says these matrices should have a determinant which is equal to a constant times Z to the power D and then inverting these matrices will lead to again a FIR matrix and this is what we need for our filter bank design procedure. Remaining question is how do you build an E of Z which is unimodular and this is also answered in this slide by means of formula here you can easily construct an E of Z by multiplying the number of matrices together and a particular sort of cascade and the cascade that you see if you read it from right to left. You see here zero and this is supposed to be a constant matrix.

So, a matrix N by N just numbers, not a function of Z. And then you multiply this with basically an identity matrix where only the last element and in the bottom right position is a Z to the -1.

And then you have a second constant matrix E1 and then again such matrix identity EZ to the -1 and then again a constant matrix and then again an identity matrix with the Z to the -1 etcetera and on you go until you have a full cascade and basically the number of matrices with this E to the -1 determine the order of the FIR matrix transfer function.

You can easily check that the determinant of such an E of Z is basically equal to the product of the determinants of the individual matrices in the cascade, and if you take a determinant of E of Zero, that will be a constant. Same thing for determinant of E1 and E2 up to El.

So these are all constants and then the determinant of such a matrix identity matrix with one Z to -1 is basically the product of its diagonal element. So that is C to the -1, so altogether the determinant of the E of Z as you see it here is going to be Z to the -L*constant, and that satisfies our definition of a unimodular matrix. So, this is how you construct A unimodular matrix. Notice that if you have constructed the E of Z like this, you can easily construct the inverse by just

inverting the cascade, basically.

And the inverse of the matrix product is the product of the inverse is in reverse order. As you see it here. You can also add delays to make everything causal and then the product of R * E will be an identity matrix. Up to this delay, which is the Z to the minus L, so it's fairly simple or easy to construct unimodular matrices and then computing the inverse. The remaining question is basically how do we construct any of Z which is then unimodular, but which also corresponds to a suitable analysis filter bank where all of the filters and each row of the EZ defines the filter of the analysis filter bank is actually you could filter could low pass filter, could band pass filter, etcetera. So if you parameterize the E of Z as you see it in this formula, the question could be what is a good setting for all these parameters such that the low pass filter in the filter bank is a good low pass filter and meeting filter specifications, etcetera.

And then the first bandpass filter again is a good bandpass filter meeting filter specifications etcetera. So you have a parameterization for the analysis filter bank as a whole and you would optimize all the unknown parameters in this E of Z cascade of fixed matrix and so you would optimize all the entries in these fixed matrices such that the low pass filter is a good low pass filter. The first band Pass Filter is a good band pass filter etcetera. So this obviously can be done. It leads to one huge optimization problem. Not easy to solve, but doable.

And so the question is, can we simplify this design procedure? Basically because this is doable but not really straightforward or easy because we're designing the full analysis filter bank in one go. All filters together in a sense, and this is something we'd like to avoid in the end.

10.10 Slide 21

Before looking into a simpler procedure on this slide and the next slide we consider an even further special case of what we had in the previous slide. And it's an interesting special case because leads to as you'll see in the next slide filter banks with interesting properties. The special case that we consider is the case in this cascade that defines the E of Z where you have the fixed matrices and the matrices with the identity matrices and the Z to -1.

If in this cascade as I've called it, if all the fixed matrices, so from right to left in the formula here the E of 0 and then the E1E2 up to El. If all these fixed matrices are in the real valued case orthogonal matrices or more generally in the complex valued case, unitary matrices. So the orthogonal version for complex matrices complex valued matrices.

Then you get a specific E of Z which corresponds to a further subsets within the subset of FIR unimodular matrices, and this subset is referred to as the FIR paraunitary matrix transfer function subsets as you also see it and the figure here.

If all the fixed matrices are orthogonal or unitary, where you remember that orthogonal matrix is a simple inverse obtained by just transposing the matrix, and the same thing for complex unitary matrices, you just take the complex conjugate transpose to compute the inverse so if you exploit this then again you can easily compute the R of Z from the E of Z and the same perfect reconstruction condition is satisfied.

If you follow this procedure so the fixed matrices the E are special matrices orthogonal unitary then the E of Z&R of Z are referred to as paraunitary matrix transfer functions and the filter banks that they correspond to the analysis and the synthesis filter bank are said to be very unitary filter banks, and the special properties of these filter banks will be discussed R attractive? and will be discussed in the next slides here also you can compare the cascade as you have it in the definition of the EZ with what we have seen in Chapter 5 with lossless lattice realizations, you will see that lossless lattice realizations are indeed special cases of such paraunitary EZ's or filter banks.

10.11 Slide 26

More unitary perfect reconstruction filter banks as we have seen them, and then in a sense also almost designed them. In the previous slide, happened to have CREATE properties, and these properties are merely listed here. The proofs are omitted and there's not really a need to try and derive these properties or prove these properties. First if the polyphase matrix E(z) is a paraunitary matrix transfer function, then the filters in analysis filter banks. So the analysis filters are or can be proven to be power complementary. So form lossless 1-inputs and outputs system of filters and so these filters satisfy this first equation where some of the squared magnitude response of all the analysis filters is equal to 1. This is an equation that we have come across and also used in chapter 5 to lossless lattice filter realization. So this is as far as the E(z) and analysis filter bank goes. If you then derive the synthesis filter bank and so the R(z) polyphase matrix from the E(z) we've seen in the previous slide. So we have a simple procedure to derive the R(z) from the E(z) and that in fact also translates into a simple procedure to derive an impulse response sequence of synthesis filter bank filter from an analysis filter bank filter. And as it's indicated here, you basically take an impulse response sequence of an analysis filter. You reverse the order of the coefficients, and you take the complex conjugate of each filter and that gives you an impulse response sequence for a filter in the synthesis filter bank. Based on this, you can also verify that magnitude response of a synthesis filter fn. Let's say the n-th filter in the synthesis filter bank is the same as the magnitude response of corresponding filter in the analysis filter banks, say hn. So the magnitude responses are the same. And then if you go back to the first equation on this slide from there you will decide that the synthesis filters also form a set of power complementary filters. So the synthesis filter bank is also a power complementary filter bank. Synthesis filters are not exactly the same as the analysis filters. You see the difference, like the equation in the middle of the slide. The magnitude responses are the same and so both filter banks are indeed power complementary filter banks.

10.12 Slide 30

Up to now, we have considered maximally decimated filter banks. The downsampling factor D is equal to the number of channels N. And we've seen in previous slides that the perfect reconstruction filter bank design and the maximum decimated case is not exactly sort of a simple task. It turns out that if we now turn to oversampled filter banks, so where the downsampling factor D is smaller than the number of channels N, the design procedure turns out to be much easier. The perfect reconstruction condition to start from, as we have seen in the previous chapter, is basically the same perfect reconstruction condition R * E is equal to an identity matrix. But now the dimensions of this equation where the dimension of the matrices and this equation are different. R is now, as we would call it, a short fat matrix D * N with a few rows and many columns and then E(z) as we would call it a tall and thin matrix with more rows and fewer columns and then a D times N matrix times N times D matrix is a D * D identity matrix and in this case so the identity matrix is D * D is a smaller matrix and the fact that it's a smaller matrix in a sense as you will see it in the next slides will lead to fewer equations in more unknowns and enhance an easier solution. The design procedure then is pretty much the same as the design procedure as we have seen it in previous slides. So first you go out and design all of the analysis filters one by one. This is basically chapter 4 where you design a low-pass filter for the analysis filter bank and then the first bandpass filter, second bandpass filter, etc. Once you have designs for all of the analysis filters, you do a polyphase decomposition of each and every analysis filter. And so each filter through its polyphase decomposition defines a row of the E(z) polyphase matrix. And then finally in step three from the of E(z), you would compute an R(z) such that perfect reconstruction condition is satisfied. And so now we consider this slightly different form from the polyphase destruction condition with different dimensions etc. If this is easy and it will turn out that this is indeed easy and then the whole procedure is in fact sort of an easy procedure. Again, we're going to consider FIR E(z), so an FIR analysis filter bank, so that for these FIR filters we can do simple polyphase decomposition to find the E(z). And with such an FIR E(z), it will indeed turn out that we can easily find the corresponding R(z) such that the perfect reconstruction condition is satisfied in most cases, except for a few contrived

cases or exceptions in a sense, as you will see in the next slide.

10.13 Slide 31

The remaining question is how you can compute the R(z) given the E(z) and the answer is basically in this slide. Assume that every entry in the E(z) matrix is an FIR filter with order equal to L_E. So E(z) is basically the analysis filter bank. Each and every analysis filter has been decomposed into polyphase components and so every entry in the E(z) is such a polyphase component and assume this polyphase component is of course again an FIR filter and the order of this polyphase component is L_E. So if the polyphase component, the entry and the E(z) matrix has order L_E, it has $L_E + 1$ coefficients as. Similarly, assume that every entry in the R(z) is again FIR filter polyphase component again with order L_R, so with $L_R + 1$ coefficients. We're computing the R(z) and the dimension of R(z) is D * N. So we have D * N entries in R of Z, and then each entry is an FIR filter with $L_R + 1$ unknown coefficients. So the total number of unknown coefficients in the R(z) matrix is indeed as you see it in yellow here D * N the number of entries times $L_R + 1$, the number of coefficients in each and every entry. Then you multiply the R matrix together with the E matrix, so R times E. When you compute this multiplication, you will have entries of the R matrix that get multiplied with entries of the E matrix, and so the R entries or L_R order polynomials and entries are L_E order polynomials. If you multiply two such polynomials together you get polynomials with order $L_E + L_R$. So with a number of coefficients, $L_E + L_R + 1$, and so every entry in R * E is a polynomial with $L_E + L_R + 1$ coefficients. And then, because of the perfect extraction condition, every such entry, the complete matrix vector product is equal to identity matrix times C to the -delta and so every entry in R * E has to be either equal to 0. So that is where all these coefficients $L_E + L_R + 1$ coefficients have to be equal to 0 or they have to be equal to the diagonal elements 1 to have the zˆ (-) and then most of the coefficients are zero except one corresponding to the zˆ (-). So every entry of the R times E basically gives you $L_E + L_R + 1$ equations in the unknown coefficients of the R(z) matrix, the total number of linear equations. There's an exercise here in terms of making these equations more explicit. The number of equations is D * D because of that being the dimension of the identity matrix in a perfect reconstruction condition. And then each entry has $L_E + L_R + 1$ coefficients that are transformed into the same number of equations. So altogether D * D * ($L_E + L_R + 1$) equations in a number of unknowns which so we have a number of unknowns, the first yellow line and then a number of equations, the second yellow line. So linear set of equations at number of equations in a number of unknowns. And this set of equations can be solved if the number of unknowns is larger or at least equal to the number of equations. And that is the first inequality as you see it here at the bottom of the slide, which can be transformed into an equivalent inequality that says L_R has to be at least equal to and then you have an expression so that is a lower bound for the order of the entries of the R(z) matrix, and this is for the further discussion in the next slide.

10.14 Slide 32

From the previous slide, we have this lower bound for the L_R. L_R has to be larger than (PPT). So assume we are given the D, the downsampling factor, given the number of channels N. You are given the L_E because you have designed the analysis filter bank and then have done the polyphase decomposition of the analysis FIR filters and hence that has defined the L_E. So everything in the right hands part of the inequality is given and then the only thing you basically have to do is pick an L_R which is sufficiently large and so the conclusion is if a L_R is made sufficiently large then you have a set of equations that can be solved and so that leads basically to the entries or the coefficients of the FIR filter in the R(z) matrix. And so the procedure as such is really simple. So you first design your analysis filter bank that defines the E(z) and from the E(z) you compute the R(z) by solving a set of linear equations that is simply enough. Special case where these equal to N So we're the downsampling factor is equal to the number of channels the maximally decimated case that we have considered first. In that case, if D=N, you see in the lower bound for the L_R and framed formula that N-D is equal to 0. So then eventually you

would decide that L_R has to be equal to infinity, which means that the filters in the synthesis filter bank would be infinitely long. So in essence infinitely long impulse response filters IIR filters, and that matches with the general statement that if you invert an FIR transfer function also an FIR matrix transfer function, you generally are in the general case end up with an IIR transfer function where then you have to worry about stability etc. So this is what we have seen in the maximum decimated case, but so not in the oversampled case. In oversampled case you compute an FIR E(z) and from there you compute an FIR R(z). As long as you have over sampling. This is sampling something that can be done if you have maximal decimation, and the case is equal to N you see the problem appearing in the lower bound for the L_R .

11 Chatper 14

Chapter 14

11.1 Slide 8

In the procedure from the previous slide, especially for the oversampled filter bank cases, very straightforward, very simple, you just go out and design each and every filter in the Analysis filter bank and that is basically Chapter 4 and that specifies the E(z) and from there you compute the R(z), etc. So the procedure is straightforward, but you could say it's rather tedious if you are hard to design each and every filter in the analysis filter bank. Imagine the number of channels is not 4 as we have in the simple example but will be 1024 or whatever. Then of course the procedure is very tedious. You wouldn't want to go out and design the first filter and then a second filter up to the 1024th filter. So where is the procedure is simple. You would like to have a more efficient procedure where as you will see it in later slides, basically you design one filter and then all of the other filters are somehow derived from this one design filter and this is where will be referred to as uniform filter banks. And then modulated for instance as you will see later filter banks. We will also consider DFT-modulated filter banks, cosine-modulated filter banks, which are interesting but not covered here, only briefly mentioned in the later slide. So we will consider DFT, so-called DFT modulated filter banks and you will see later slides what we mean by that. And again we're going to consider the maximally decimated case, so maximally decimated DFT modulated filter banks and then again the oversampled DFT modulate filter banks case. So we first have to define what we mean by uniform filter banks and then see how or where the name modulated filter banks comes from.

11.2 Slide 9

The rest of the chapter we're again going to consider maximally decimated as we'll see DFT modulated filter banks first and then and then later oversampled DFT modulated filter banks. In this slide we first define uniform filter banks versus non-uniform filter banks and we're considering analysis filter banks first and by means of a simple example here with the four-channel filter bank, it's immediately clear that what we mean by uniform and non-uniform case. The non-uniform case, everything that is not uniform is referred to as non uniform. And this is the case where the different filters in the filter bank could have different responses or differently looking responses with different ripples in the passband or whatever. Even different bandwidths could be something you can have. So that's the sort of easy definition of non-uniform. On the other hand, uniform filter banks are filter banks where all of the filters do the same as we see here. In the frequency domain or when you consider the frequency domain representation or shifted versions of each other, we have a defining equation here. The framed equation on the slide here where we say in a filter bank the n-th filter and so we're talking the analysis filter bank. So the n-th analysis filter is equal to the analysis filter H0. So that is basically the lowpass filter in the analysis filter bank where z is replaced by zeˆ(-j 2 pi n/N), where N is the number of channels in the in the filter bank.

The meaning of this equation is that if you do this multiplication and evaluate that on the unit circle to get the frequency response. So then you replace z by e ^ (j) and multiplying by e^ (-j 2 pi n / N) Then you would basically have H0 e^(-j)and then –j 2 pi n / N and so each Omega is basically shifted over an amount corresponding to the 2n/N and so the end result of this is basically that you tip the frequency response of the H0 and you shift it over an amount. And this is also what you see in the figure here for the first figure for the uniform filter bank. So you take your H0 and this is your low pass filter and you shift it over an amount over the frequency axis shifted to the right over an amount. And this simple four general case which is the amount where the amount is equal to 2pi/4 = pi/2, and that keeps the frequency response for H1, the second filter in your analysis filter bank, if you continue shifting the H1 over the same amount you end up with H2 and when you shift H2 over the same amount you end up with H3 and that is basically the four filters in your 4 channel analysis filter bank. If you want to continue shifting the H3 over the same amount by over 2 then the center frequency would land in 2pi which is where 2pi is equivalent to 0. So then you're back at H0 to low pass filter. So this is the defining equation that you have here in the frequency domain. This is equivalent to a time domain expression where you can say that the impulse response sequence for the nth filter and the analysis filter bank is the impulse response sequence of the H0 filter modulated with the complex exponential function as you see it here. And this modulation indeed in the frequency domain leads to the frequency shift as we have it in the defined equation. The H0 to low pass filter is then referred to as a prototype or prototype filter. And as you will see in later slides, this is basically the only filter that we will actually design. So the design procedure will be limited to designing a good prototype filter. So a good lowpass filter H0 and then all of the other filters are derived from this designed lowpass filter by shifting the H0 frequency response over the frequency axis as we have discussed it.

11.3 Slide 11

It's significant advantage of uniform filter banks is that they can be realized extremely cheaply based on polyphase decompositions on the one hand and then a DFT matrix or a DFT operation which in practice will be realized or implemented by means of an FFT algorithm. And because of the usage or the appearance of the DFT matrix, these filter banks are then in the referred to as the left modulated filter banks. To derive this cheap realization of the uniform filter bank, let us start from a filter bank with prototype filter H0 and then H1 and H2 etc, derived from a prototype through the defining equation as we had on the previous slide. So the nth filter is basically the prototype filter where z is replaced by z * e (- j...), which corresponds to the shifting over the frequency axis. Now assume that the prototype filter is decomposed into its polyphase decompositions, so that is the second equation on the slide. Here H0 is equal to a summation over basically the n polyphase components, the En(z ^N) and then multiplied with the delay filter for each polyphase component z^(-0) for the first component and then z^(-1) for the second component, etc. So this is the usual definition for polyphase decomposition applied to the prototype filter H0 and the polyphase components are denoted here as the E_n. Now if you then make an attempt to derive a polyphase decomposition for the nth filter, you have to do a bit of a derivation and I'll now go into the details, but basically you start from the defining equation or the end filter and this is the one but last equation on the slide. Here the end filter is equal to the prototype filter with z replaced by z * e ^(-j...). And this is where you can substitute the polyphase decomposition for the prototype filter, again with z replaced by z * e ^(-j...), and you do a bit of a simple derivation. The end result is that you end up with summation, which looks like the summation in the polyphase decomposition for the prototype filter, with exactly the same polyphase components. So the conclusion is (underlying with the white bars) polyphase components E_n. So conclusion from here is that basically from the polyphase decomposition of the prototype filter, you can also immediately write down a polyphase decomposition for the nth filter, which has basically the same polyphase components. The only difference is basically additional multipliers or factors as you see them appearing in the last equation where you have W to a power and then something like -N times another n over bar where the W is a factor which is defined here to be E to the minus J 2 divided by the capital N, number of channels as you have in the filter back. So the from the polyphase decomposition of the prototype filter, immediately you have the

polyphase decomposition for any filter in the filter bank. The only thing you have to plug in is sort of all of the powers of W have where you use these powers of the W additional factors in the polyphase decomposition.

11.4 Slide 12

In the previous slide, we have expression for the polyphase decomposition of each and every filter in the analysis filter banks so H0 H1 H2 up to HN -1 and if we put all these equations together in one large equation then you end up with what you have here given in this slide. In this equation, for the time being, ignore U(z) and the left hand side of the equation and then also the U(z) in the right hand side of the equation, which you then see is the left hand side of the equation of vector with all of the analysis Filter banks filter transfer functions H0 in the first position and then H1 H2 up to HN-1 in the last position and then what you see is a matrix times a vector, where in the vector you have the polyphase components of the prototype filter E0 in the first position, E1 in the second position E2 up to N -1 and as you have it in a polyphase decomposition all of these transfer functions are a function of zˆN and then they also get multiplied by delay filters zˆ0 in the first position and then zˆ-1 in the second position, zˆ-2 up to zˆ-(N – 1). That is the vector as you have it in the right hand side of the equation and this gets multiplied by a matrix. In the matrix you see powers of this defined factor W to the zero which is equal to 1, and then again W to the 0 which is equal to 1 all once in the first row. And if this row of once multiplies the vector with the polyphase components. That indeed gives you the definition of a polyphase decomposition of the prototype filters. So H0 would be equal to 1 * E0 + 1 * zˆ- 1 * E1 + 1 * zˆ-2 * E2, etc. And then the 2nd row of the matrix has different powers of the W and that corresponds to the polyphase decomposition of the H1. So H1 is then going to be W to the 0 * E0 + W ˆ-1* zˆ-1*E1, etc. And so the equations in the first in the previous slides were basically every such equation fills a row of the matrix and leads to the overall equation as you have it here. The matrix that you see appearing where every entry is a power of this W factor is obviously a DFT matrix, or to be more specific and inverse DFT matrix up to a factor which is the M here, which later actually will be removed, sort of absorb by polyphase components. So basically this is an inverse DFT matrix. And now the meaning of this is basically as follows if you're now indeed insert U(z) in the left hand side of the equation and then also U(z), so post multiplied by U(z) and the right hand side of the equation. The formula basically reads as follows. In the left-hand side of the equation you have the vector with all of the analysis filters H0 H 1H2 etc which multiply U(z). So you have an input signal and in the Z domain U(z) which gets multiplied by H0 and then H1 etc. And that represents the filtering of the input signal with all of the filters in the analysis filter bank. And then the equation says that in order to obtain all these filtered versions of the input signal, you could realize this in an alternative fashion as follows. And then you go to the right-hand side of the equation and you read it from right to left. You take the same input signal U(z), and then you filter it with all of the polyphase components off the prototype filter that you have E0 E1 E2...and multiply with the delay filters zˆ0, zˆ-1, zˆ-2... And then the result is multiplied by this inverse DFT matrix and it will turn out especially when we substitute cheap algorithm for the multiplication with the inverse DFT matrix, the inverse FFT matrix, etc that this indeed leads to a very cheap realization as you will see it in the next slide.

11.5 Slide 13

Graphical representation by means of a block scheme of the formula as you had in the previous slide is provided in this slide and you should actually make sure that you thoroughly understand or see that this block scheme indeed corresponds to the formula as we added in the previous slide. In this block scheme you see to the left and input signal which is U[k] here time domain representation. To the left of the block scheme, you have to recognize that this input signal first gets filtered by E0 and then in the next branch gets filtered by zˆ-1 because the one delay element then E1, and then in the next branch it gets filtered by zˆ -2 because two delay elements on the way to E2 and then gets filtered by E2 and then finally in the lower branch you would have zˆ -3 times the input signal because of the three delay elements on the way to E3

and then gets filtered by E3 and this intermediate result is then multiplied by the inverse the left matrix, the F inverse to produce and the symbol for channel example of course to produce 4 output signals which corresponds to the input signal filtered by H1H2 and filtered by H3. This is a very cheap realization. You should realize that the polyphase components E0E1E2E3. Altogether you have four filters in a sense, but the complexity, the number of coefficients you have in all these polyphase components together is equal to the number of coefficients you have in the prototype filter. So the complexity of running these four filters in this case is equal to the complexity of running just one filter, which is the H0. And then if you consider the multiplication with Fˆ-1 the inverse DFT operation as a cheap additional operation because we are going to substitute with FFT or inverse FFT algorithm of course. Then basically you can say that is an important statement that the realization complexity or implementation complexity of the uniform filter bank, DFT Modulated analysis filter bank. The implementation complexity of this is basically your corresponds to the implementation complexity of just one filter, the prototype filter, which is remarkable and actually a spectacular statement. So you make a complete filter bank, in this case with four filters in the filter bank, and you realize or implement it with the complexity that corresponds to the complexity of only one filter and a four-channel case filter bank, this is perhaps not so spectacular, but think of a 1024 channel filter bank. You would derive or implement a complete filter bank with 1024 filters. At the complexity, implementation or realization complexity of basically just one filter, that is quite spectacular. So this is an extremely attractive cheap realization of the DFT modulated uniform filter bank. So a uniform filter bank like DFT modulated filter bank is attractive. You design one prototype filter and then all of the other filters are derived from there. So it reduces the design complexity. You design only one filter, and it also reduces the implementation or realization complexity because basically the realization complexity is the complexity of realizing only one filter basically the prototype filter.

Sldie 14

The quick remark here in special cases the case where all of the polyphase filters are components are equal to one, as you see it in the graphical representation in the block scheme that basically all the polyphase components are void or can be left out what remains as the F inverse and this is something that we have come across. It is something that has been referred to as the DFT filter bank in the previous chapter, with the magnitude responsibility for the analysis filters and a simple 4 channel case. So the DFT filter bank is a special case, the simplest case of a DfT modulated filter bank, as we're studying it in this chapter.

11.6 Slide 15

Final remarks here in terms of efficient realization, the block scheme as we added in the previous slide is an analysis filter bank which is then followed by down sampling operations. So in this case we're still considering maximum dissemination. So if you have a simple example for channel analysis filter bank, this will be followed by four-fold downsampling. General case you would have n-channel filter bank followed by n-channel and fold down sampling now with the realization in the previous slide with polyphase components so we've often used because you can swap them with up and down sampling operations. Same thing here. So we have to filter with polyphase components. This multiplication with matrix F inverse is merely multiplication of signals with fixed coefficients, the entries in the F inverse matrix, and then summation operations which are also easily or straightforwardly swapped with the downsampling operation. So F inverse can first be swapped with the downsampling operation and then the filtering with polyphase components can also be swapped with the downsampling operation. So then you would have the downsampling up front and then after the down sampling you have filtering with polyphase components where zˆ4 has been replaced by z. Remember to normal identities and then after the filtering with all of the polyphase components you have the multiplication with F inverse. And here again it is restated that this is a very efficient realization. All of the operations the filtering with polyphase components that multiplication with inverse DFT matrix. All of this is done at the low sampling frequency side of the down sampling, so your first down sample goes to a lower sampling frequency, and then I'll only have all of the operations, the filtering and the multiplication with

the F inverse, so this is extremely efficient.

11.7 Slide 16

Everything in the previous slides applies to the analysis filter bank, then the obvious question is can you do the same development for the synthesis filter bank? And the equally obvious answer is yes. So in a similar fashion as we have defined uniform analysis filter banks, we can first define uniform synthesis filter banks and the defining equation is given in this slide first formula. In this slide different from the defining equation in the analysis filter bank case in that we have an additional phase factor which is added merely for convenience with this additional phase factor. It turns out that all later formulas are simple in a sense that an inverse DFT matrix is replaced by a DFT matrix etc. So this is the reason for inserting this additional phase factor. But basically we have a synthesis filter bank with the prototype filter F0. This is going to be a low pass filter again, and then the NTH filter in the synthesis filter bank is derived from prototype filter where z is replaced by z*eˆ(-j 2 n/ N) as we have seen it in the defining the uniform analysis filter bank and the multiplication with the eˆ(-j.…) in response to a shift over the frequency axis again. And then this gets multiplied by this additional phase factor. Don't ask questions about the additional phase factor, as I said, it's merely introduced for convenience. Then again, if you start from the polyphase decomposition of the prototype filter F0 with polyphase components Rn. From there you can drive the polyphase decomposition for the NTH filter and the synthesis filter bank and it will basically have the same polyphase components same with the white underlining Rn, but then with additional factors, where the additional factors are the powers of this same quantity W.

11.8 Slide 17

Similar to what we had for the analysis filter bank case, you can then again merge all the polyphase decomposition equations from the previous slide into one large equation and then rather than an IDFT matrix, you would see a DFT matrix appearing in the equation and that leads to a realization or block scheme as you see it here. Basically you would to the left start with the sub band signals which are then multiplied. So a collection of samples of the sub band signals is a vector that gets multiplied by the DFT matrix, the F matrix. The result is then filtered by means of the polyphase components of the prototype filter. And now minor difference here is that polyphase components are stacked from bottom to top. So if you read block scheme from bottom to top, you would see R0 first and then you go from bottom to top R1, R2, R3 for this simple four channel case and then the resulting signals filtered by the smaller phase components enter into a delay line and a summation operation to provide you with the output signal. So you can easily verify for yourself that this indeed completes to or corresponds to the complete operation of the uniform. A synthesis filter bank derived from a prototype filter, of which the polyphase components are given as R0, R1, R2, R3 is very simple for channel case.

11.9 Slide 18

And then finally, again, as we know that the synthesis filter bank operations are always preceded by an up sampling operation and the simple case here, for channel case maximal decimation, so four fold up sampling again, you can swap the up sampling operation with the multiplication with F and then with the filtering operations with polyphase components where then you would have at the bottom of the slide first the multiplication with F, then the filtering with polyphase components where the z4 has been replaced by z and then only at the end you would up sample. So go back to the higher sampling frequency and then merge the output signal by having the chain with the additions and the delay operations. So that again leads to a very efficient realization of a uniform. A DFT modulated synthesis filter bank.

11.10 Slide 19

Now back to perfect reconstruction for maximally decimated DFT modulated filter banks. If you take the synthesis filter bank from a previous slide and you glue it together with the analysis filter bank from slides 17, you obtain the blocks scheme as you see it in this slide. Still for that simple case where small example where the number of channels is equal to four, but obviously this is straightforwardly generalized for the general case and it different from four. So you see the analysis filter bank that defines the boldface E(z) matrix, which is then basically, as you see it here in the first formula to the right is basically the F inverse matrix and then multiplied by a diagonal matrix where the diagonal elements or the E polyphase components and you have a similar expression for the synthesis filter bank boldface R matrix and so starting from our perfect reconstruction condition that the product of the R and E is equal to an identity matrix up to a delay, you can easily, from the formulas and also from the block scheme, decide that basically the poly phase components in the synthesis filter bank have to be the inverse of polyphase components in the analysis filter bank polyphase components of the prototypes of the analysis and the synthesis filter bank. So if you look into the block scheme, for instance, you immediately see that the F inverse times F is an identity matrix. So basically that part drops out and then in order to have perfect extraction, then R3 has to be the inverse of E0. And R2 has to be the inverse of E1 and R1 has to be the inverse of E2 and R0 has to be the inverse of E3 all up to delay. So we have a very simple result here or condition for perfect reconstruction, basically. So the polyphase components of the prototype filter for the synthesis filter bank in reverse order. Basically check the block scheme or equal to inverse up to a delay of the polyphase components of the prototype filter for the analysis filter bank, and this will lead to a symbol design scheme, as you'll see in the next slide.

11.11 Slide 20

From here, the design procedure looks very simple. You first go out and design the prototype analysis filter, the H0, which would be a low pass filter. So this is where you apply chapter four to design a good low pass filter H0. And here again we're going to use FIR filter. So we design H0 to be an FIR filter so that we can immediately decompose it into its polyphase components in step two, and so this is where we have the E polyphase components and then finally in step three, we would only have to invert the E polyphase components of the prototype analysis filter to obtain the polyphase components of the prototype synthesis filter, and we added delay z to the minus delta for after or with all the inversions. For instance, to make things causal or whatever, and an issue here is that we're inverting again scalar FIR transfer functions and in the previous slide we have seen that if you invert a scalar FIR transfer function, you obtain, in general, an IIR transfer function where we should be worried about stability because zeros if the originally transfer function which can safely lie outside the unit circle or converted into poles, but then have to lie inside the unit circle to guarantee stability. In a previous slide, we escaped from this problem by then considering matrix transfer functions FIR matrix transfer functions which can have FIR inverses. But here basically there's no escape because we're indeed considering scalar transfer functions and then the inverse of these scalar transfer functions.

11.12 Slide 21

Whereas the design procedure in the previous slide is conceptually very simple, it will turn out that it doesn't really leave much design freedom basically for the analysis filter bank prototype H0, following a similar development as we had it earlier for the general case. So when we were looking not yet into the DFT modulated filter banks, you could ask can we design an FIR boldface E(z) matrix such that it inverse boldface R(z) matrix is, again, an FIR matrix, but it will turn out that in this case where the E(z) is factorized as you had in the previous slide as basically the F inverse, inverse DFT matrix times a diagonal matrix with FIR functions on the diagonal. Then there is not much left and that is also indicated in the figure at

the bottom of the slide, so we were looking into E(z) transfer functions and then a subset of those namely the FIR E(z) transfer functions. And then again a subset of this subset, namely the unimodular FIR E(z) transfer functions. But now we're also looking into a subset of E(z) that are factorized as F inverse times diagonal matrix. And it turns out that the intersection of this last subset and the subset of unimodular FIR E(z) is restricted to transfer functions where all of the FIR transfer functions on the diagonal. So the En polyphase components are themselves unimodular and so that basically means that each En is unimodular. So in itself is equal to a constant times a delay filter. And obviously if you were to invert such a polyphase component as you added in the previous slide, then that indeed leads to an FIR polyphase component, R polyphase components as you had in the previous slide. So it means that you are to design an H0 analysis filter bank prototype filter where all of its polyphase components have to take this simple form. Every polyphase component has to be a constant times a delay and it will turn out that this significantly reduces the design freedom for the analysis filter bank prototype filter. A useful example, though this is the case where all of the polyphase components of the analysis filter band prototype filter H0 or constants indicated here as Wn and this leads to something which is referred to as a sort of a window, IDFT DFT filter bank. So you basically have an IDFT or DFT operation and the analysis and synthesis filter bank up to a window function and that will also be referred to as the short time Fourier transform based filter bank, which will be studied in the next chapter.

11.13 Slide 22

In the general case we further looked into a subset of unimodular FIR transfer functions, namely paraunitary FIR transfer functions which turned out to have interesting property. So, here again you could also take it one step further and ask can we have a paraunitary FIR E(z) with hence also a simple inverse and interesting properties but similar to what we had in a previous slide. Here you would conclude that for the boldface E(z) to be paraunitary, then each polyphase components the E polyphase components of the analysis filter bank prototype filter. Each such polyphase component then in itself has to be paraunitary and a polyphase component is a scalar transfer function. A scalar transfer function is paraunitary when it is all passed. And then when on top of this it has to be an FIR function. Then the only thing that remains is where basically a polyphase component is either plus or minus one times a delay function. And if you leave out the delay function and you take plus ones everywhere, then you're back at the IDFT DFT filter bank as we have seen it in the previous slide. So basically with this you would obtain only trivial modifications of the IDFT DFT filter bank, which are not much more interesting than the IDFT DFT filter bank itself. So again, we do not have much design freedom left based on the design procedure from two slides back.

11.14 Slide 23

The conclusion from the previous slides is that DFT modulated filter banks are great in the sense that you would have to design only prototype filters rather than a full filter bank. And also from a realization and implementation complexity point of view, they're very attractive. But then if we stick to a maximum decimation, then there's not much design freedom for the analysis filter bank prototype filter left. And that is bad news. The good news is that you can generate more design freedom by moving over from maximally decimated filter banks to over samples. Then again DFT modulated filter banks and this will be discussed in the rest of the chapter. Another alternative is to use what is referred to as cosine modulated filter banks and this is not covered here, but they're interesting filter banks where you can have similar advantages is what we have mentioned for DFT modulated filter banks, but then still for under maximal decimation, have sufficient design freedom to design useful filter banks. So here you do not even have to give up on maximal decimation. Cosine modulated filter banks are drived in a bit of a different fashion compared to the way we drive or constructed DFT modulated filter banks. It's indicated in the figure here. Starting point is a prototype low pass filter, which is denoted here as P0 with a cut

off frequency, which is plus and minus pi/2N rather than divided by N as you would have it for H0 in the DFT modulated filter bank. So, you see that the bandwidth of the P0 is only half the bandwidth of the H0 as we had it in the DFT modulated filter bank and then you take the P0 and you shift it over the frequency axis over an amount to the right and at the same time you also shifted over the same amount to the left and the sum of these two shifted versions of P0. Sum of these two terms in a sense then corresponds to the H0 LP filter for instance an analysis filter bank as you see it in the figure here if you shift the P0 further up to the right and then also further up to the left again you have two terms and these together define H1, then the first band best filter of your filter bank and on you go to construct H2, H3 etc. What you also see is that because of the construction mechanism, as you have it here, you preserve symmetry in the frequency domain, which corresponds to having real valued coefficients for the filters and also or equivalently stated real valued impulse responses for the filters, which was not the case in the DFT modulated filter banks. So you have a simple construction mechanism. It will turn out if you express all this by means of formulas, equations then compared to DFT modulated filter banks rather than DFT matrices, you would have discrete cosine, transform matrices etcetera. Everything is very similar but we're not going to go into further details. Nevertheless, again with cosine module like filter banks, you can have sufficient design freedom for perfect reconstruction under still maximal decimation, which is very interesting.

11.15 Slide 25

So let's now look into oversampled DFT modulated filter banks. So where the down sampling and up sampling factor D is smaller than N, the number of channels in this case. As you see them and the figure here copied from previous chapter you would have a boldface E(z). You see to the four matrices, which is it's all thin matrix, remember, N by D, so with more rows than columns, and then similarly in the synthesis filter bank, you would have a boldface R matrix which is short and fat, so with more columns than rows, D by N. Now in a maximally decimated DFT modulated filter banks we had a specific factorization for the E and R matrix, which was given in the previous slide. So the E matrix is the IDFT matrix times a diagonal matrix with the polyphase components of the analysis filter bank prototype filter on the diagonal in the similar expression for the R matrix. And in this or in these equations, all matrices that you see are N by N matrices. For the over sampled case, over sampled DFT modulated filter banks. What we're going to see in later slides by means of examples is that the E(z) matrix satisfies a similar factorization where E(z) is again equal to an inverse DFT matrix. So, this is N by N and then a B(z) matrix which again is tall and thin, similar to the E(z) matrix. So, and by D more rows than columns. But now rather than in a previous case B(z) being a diagonal matrix, B(z) in this case is going to be a structured and sparse matrix. Diagonal matrix is obviously also a structured and sparse matrix, but here we're generalizing this to all thin matrix and you will see specific structures for the B(z) matrix in later examples. And similarly for the R matrix, the R matrix will satisfy factorization with the F matrix, DFT matrix, and then a short fat D by N matrix C(z), which again is going to be structured and sparse. We're not going to display the full theory for the general case. For general D and Ns. We're going to focus on two examples for specific instances for the N and the D and then see what it's like and from there it will turn out that you can immediately generalize for any other instance for the N and the D.

11.16 Slide 26

Let consider the first example. Now we are going to design an over sampled DFT modulate filter bank where the number of channels is the N is equal to eight, so a channel filter bank and the decimation factor will be smaller than eight will be equal to four so this is in a sense two fold oversampled filter bank. The design procedure will start by designing the prototype analysis filter, H0 will be a low pass filter with a bandwidth which is basically equal to two by the full range divided by eight, because we're going to have an eight channel filter bank, so we design a low pass prototype filter for the analysis filter bank and then similar to what we had in a maximally decimated case, we're going to use the polyphase

decomposition of this H0 matrix. The difference here is that we're not going to use immediately an N fold polyphase decomposition. We're going to use N prime fault polyphase decomposition where N prime is defined here in the first formula and you should not try and understand this this formula it is just given here and it will turn out that in this example and also in the next example things come will come out nicely if you use that type of polyphase decomposition and then in general for any other instance for N and D will turn out that by using this polyphase decomposition. This N prime fold polyphase decomposition things indeed will always come out nicely. So, we have this magic formula for N prime, N prime is equal to N times D divided by the greatest common divisor of N and D, so in this case N is equal to eight, D is equal to four and then the greatest common divisor of eight and four is four, so altogether we have eight times four divided by four is equal to eight, so for this case N prime is equal to the number of channels eight but that will not be a general statement. So, N prime is equal to eight which means we have to do an eight fold polyphase decomposition of H0. And this is specified here with the a E polyphase components.

11.17 Slide 27

With these obtained polyphase components for the analysis filter bank prototype filter, we now going to construct this sparse B matrix and so in this case B is all thin matrix with eight rows and four columns and it will be constructed as you see it in the first formula in the slide here. So we're basically filling the diagonal in this B matrix. So, if you start with the one one entry in the matrix, this is where you put the first polyphase component E0 and then basically you follow the first diagonal or the main diagonal. So you go to the two two entry. This is where you plug in E1 and then the three three entry plug in E2 and then the four four entry plug in E3. If you were to continue on that diagonal, this is where you would leave the matrix to the right and then you go back in to the left, sort of a similar mechanism as you have seen it in circular matrices. So you go back in to the left and in the fifth row, first column, and this is where you use the next polyphase component, which is E4r, but this is where you also insert an additional delay operation z-4, where this four corresponds to the down sampling factor, the four fault down sampling factor, and then again you run down this diagonal, so you go to the next diagonal elements plug in E5 again with z-4, and then next diagonal element E6 again multiplied by z-4. And then the final diagonal elements E7 again multiplied by z-4. Do not ask questions. Just do it like this and you will see a more general version of this mechanism. This is the way we construct this structured sparse salt(?) in B(z) matrix. If you use that in the block scheme for the analysis filter bank, as you see it here, your input signal would first go into the delay line as you always have it. And then the boldface E(z) matrix which is where which consists of first the B matrix to all thin. And then the square F inverse matrix, which in this case is eight by eight, so that gives you your eight output channels, which should correspond to, first output should be input signal filtered by H0 and the second output should be the same input signal filtered by H1 etcetera up until the last signal, which is the input signal you filtered by H7 and it's actually easily proved that what you have here is indeed a DFT modulated filter bank and this is in the the equations at the bottom of the slide. So if you start from the left here, left hand part of the equation you would see the U(z) which is the input signal, read that part from right to left. Again, first gets multiplied with the delay operations 1, z-1, z-2, z-3, then gets multiplied by the B matrix, then gets multiplied by the F inverse plug in the previous expression for the B matrix and then verify that indeed that leads to a DFT modulated filter bank derived from the prototype filter with the E polyphase decomposition. So, this is indeed a valid DFT modulated filter bank constructed based on an IDFT matrix F inverse eight by eight for a channel filter bank and then it's all thin B matrix eight by four, which is filled with the polyphase components of the analysis filter bank prototype filter.

11.18 Slide 28

The analysis filter bank is again followed by a decimation operation. So, in this case a four fold decimation. Again we're going to use the multiplied density(?) to swap the down sampling operations with the multiplications with the F inverse and the B matrix. And the result is indicated here. So, this is where the B(z) to the four is turned into a B(z) to the one because of the noble identities(??). So, this is our efficient realization of the analysis filter bank so all of the operation again happens at the low sampling frequency side of the down sampling operation and basically the realization or implementation complexity of that DFT modulated filter bank, if you ignore the complexity of the IDFT operation, corresponds to the complexity of, basically realizing the prototype filter only by means of its polyphase decomposition, so again an extremely efficient realization for this DFT modulated filter bank.

11.19 Slide 29

The synthesis filter bank is similarly constructed. Block scheme is provided in the slide here with the F matrix rather than the F inverse now, and then the structured sparse matrix C(z) matrix. And then finally the up sampling and summation in the delay, the structured sparse C(z) matrix is now formed in a bit of a different fashion. It's again formed based on the Polyphase components of the prototype filter for the synthesis filter bank, but it's filled in reverse order. So with the first polyphase components R(z), you start filling the matrix basically at the end. So you go to the bottom right element in the last row in the last column and this is where you plug in R0 and then you go up this diagonal and put the R1 and then R2 and R3. This is where you leave the matrix at the top and you go back in at the bottom after having added a z-4, which because of the four fold down sampling, the swapping is now turned into a z-1 and then R4 and then again you move up to diagonal with R5, R6, R7 and each time multiplied by the z-1.

11.20 Slide 30

You take the analysis filter bank block scheme glued together with this synthesis filter bank block scheme. Perfect reconstruction condition is again you multiply the R and the E together and that should be equal to identity matrix up to a delay.

11.21 Slide 31

The perfect reconstruction leads to a simple matrix equation with only the C matrix and B matrix. Multiplication of C and B has to be equal to an identity matrix up to a delay. Now if you look into the structure of the C and B matrix you can easily check that all of the off diagonal elements of product C * B are immediately zero. These corresponding equations are basically void. And so you only have to look at the diagonal elements in this matrix equation. So the right hand side part you would look into the diagonal elements of C * B and the left hand part these have to be equal to the diagonal elements of the identity matrix which are all equal to 1 times z to the minus something. If you plug in again the formulas for the C and B you would see those equalities or equations corresponding to the four elements in the case where delay is z to the -1 minus, 4 polynomial equations as you see them here. So each polynomial equation corresponds to 1 diagonal element of the 4 by 4 identity matrix Here you can analysis similar to what we had in the general case and slide 11 actually if you assume that all of the E polyphase components or FIR filters with order LE and if you assume that all of the R polyphase components or FIR filters with order LR then each such polynomial equation basically corresponds to at least one equations and a number of unknowns where the number of unknowns is 2(LR+1). Again you can decide that this can

be solved you will have more unknowns than equations in the case that LR is at least equal to LE - 1 so again if you use this sufficiently large LR so each of these four equations you can solve or compute.)

11.22 Slide 32

With this the design procedure for perfect reconstruction DFT modulated filter bank especially case here with 8 channels and 4 fold up and down sampling. First design the prototype filter for the analysis filter bank so with a bandwidth of 2pi divided by 8 because it's an 8 channel filter bank. You design a FIR H0(z) based on chapter 4 and then you do polyphasic decomposition. Then finally as you have seen in the previous slide you compute pairs of polyphase components of the synthesis filter bank from pairs of polyphase components of the analysis filter bank prototype and that leads to set some equations that can be solved. At this point it's relevant to compare that to maximum decimated filter bank so if you were to have an 8 channel filter bank in this case with eight fold I should say up and down sampling then rather than pairs of R that are computed from pairs of E, you would have one R that is computed from one E. This is where you would have to invert the E polyphase components. Here because of the fact that you have oversampled 2 fold oversampling basically fourfold up and down sampling for a 8 channel filter bank, you compute R from E that is something which is possible without explicit inversion of FIR transfer functions and that is truly what makes it all possible and then give you sufficient design freedom.

11.23 Slide 35

Second example that we are going to construct the filter bank with N is equal to 6 so 6 channel filter bank where the decimation is equal to 4 so that corresponds to non-integer oversampling in the sense. So the first step will be to design the analysis is filter bank prototype H0 and this will be an FIR filter again and that we do only phase decomposition. Here again we have to use this magic formula that says what type of polyphase decomposition we actually have to use and prime is again equal to N times D divided by the greatest common divisor of N and D. We have to do a 12 fold polyphase decomposition so 12 polyphase components which are the E functions as you see equation here.

11.24 Slide 36

The only thing that is different here is the way we have to construct the structures and Be matrix. So the B matrix will be filled with the 12 polyphase components and the way we filled this is indicated in this slide and it is sort of representative of the general procedure for constructing this structure. Basically you have to follow the red line which is added here. you always start with the top left element of the matrix. Dimensions of the matrix is 6 by 4 in this case 6 rows because of the six channels and 4 columns because of the fourfold down sampling. So 6 by 4 matrix you start in the top left position and this is where you plug in E0 first polyphase components. Then you sort of run down the diagonal into (2,2) position you plug in E1 in the (3,3) position you plug in E2. If you continue on the same diagonal of course this is where you leave the matrix to the right this is where you have to jump to the left and go back into the to left in the next row. So in the next row the first column you would see the E4 elements which because whenever you jump from right to left you also have to add the delay operation which in this case is Z to the -4 because fourfold down sampling. So this is where you have E4 and then if you continue on the diagonal, you leave the matrix at the bottom this is where you have to jump to the top of the matrix and continue in the next column at the top of the matrix. So in the first row third column this is where you enter E6 and E6 continues with the same delays-4. Run down the diagonal again if you were to continue on the same diagonal again you would leave that matrix to the right jump back in the left with the next polyphase component E8 but whenever you jump from right to left you have to add z to -4 so z -4 * z - 4 is z-8 as you see it here. And then again

you further continue on the same diagonal. To check if this indeed corresponds to the DFT modulated filter back or for instance the first output from the F-1 matrix would be the input signal filtered by H0 and the 2nd output would be the same input signal filter by H1 which is the shifted version of the H0, you follow the equations at the bottom of the slide. That actually in essence response to components polyphase components of a sixfold polyphase composition rather than 12 fold polyphase decomposition which then gets multiplied by F inverse and that indeed corresponds to the six channel DFT modulated filter bank.

11.25 Slide 37-38

Everything is basically the same as we had in the previous example. You can use the noble identity to swap the fourfold down sampling first with the F inverse and then with B(z4) and this becomes a B(z) matrix where the Z-4 delays perhaps or turn it to Z-1 delays you do a similar construction for the synthesis filter bank. So similar construction for the c(z) and then perfectly destruction conditions again you would multiply the E(z) and the R(z) said that equal to an identity matrix and that leads to polynomial equations that will allow you to compute the coefficients of the synthesis prototype filter polyphase components from the polyphase components of the analysis prototype filter.