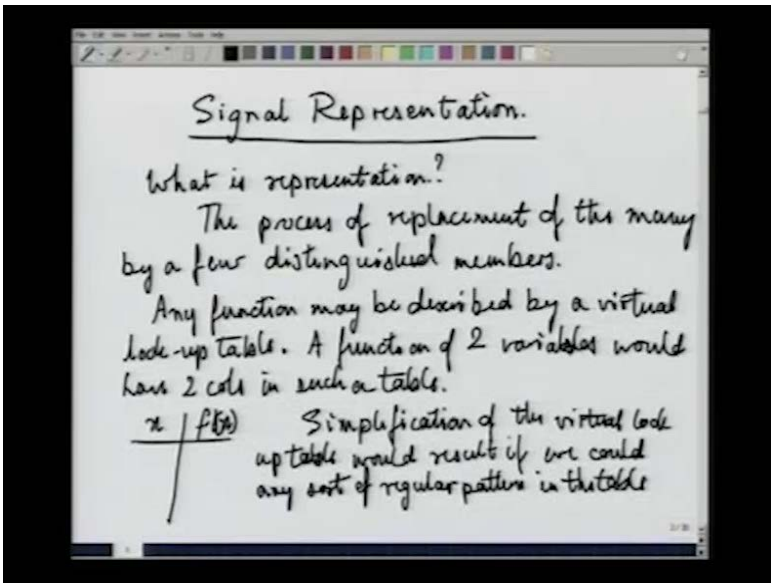
Signals and Systems
Prof. K. S. Venkatesh
Department of Electrical Engineering
Indian Institute of Technology, Kanpur

Lecture - 24
Theory of Signal Representation

(Refer Slide Time: 00:33)



What we are about to begin to discuss is called the theory of signal representation. What is representation? The term representation arrives and is encountered in so many different contexts from politics to signal processing. So, what is representation in general? It is about reducing the number of objects that need to be encountered that is to say if the objects in question are signals. Then what we seek by employing a representation is a means of dealing with only a small number of signals rather than all the signals that are theoretically possible.

If the objects are people, then representation is about a small number of people taking decisions, taking actions on the basis or on behalf of a large number of people. So, that is representation is general. Representation can be set to be the process of replacement of the many by a few distinguished members. So, that is a satisfactory definition of what representation means in the most general context. Now, focusing our attention upon system theory signals and system theory, what is the need for representation here?

And how do we go about bringing up or constructing a representation of signals. Let us understand what we gain by using a representation in the context of signals and systems. And what features, what advantages having a representation will yield to us. All right, suppose you wanted to have a description of any function. This has already been mentioned in certain earlier lectures of this course. If you want to thoroughly describe any function and let us say in this context that that function is a signal, it is let say it is a function of time.

Then I had said that you can always represent the function as a virtual lookup table. Any function may be described by a virtual lookup table All right. Now in general a function of one variable will have two columns in this virtual table. The left column would carry a list of all possible values that the independent variable can take and the right column would carry the corresponding values that the function takes against each value of the independent variable.

So, you would have a table that looks like this, values of x on this side and values f of x on this side this is what I mean by a lookup table. I call it a virtual lookup table, because while in the real world around us a table has a finite number of rows. This virtual lookup table can possibly have an infinite countably or uncountably infinite number of rows. Because there might be that many possible different values for x and for each of those values affects f of x needs to be specified.

So, we have a virtual lookup table. A virtual lookup table is possibly infinite in size possibly even uncountably infinite in size, but it is a very useful concept to play with. Now a general function be it a signal which maps voltages to time or rather which maps time to voltages or it is a system it maps signals to signals in all these cases a function is a lookup table a virtual lookup table.

Now, if you want to comprehend the information in the table completely. So, that you may put it to use to predict what output will result for the function or for that system, when the input is specified if you want to do this you normally have to simply memorize the entire table. Memorizing the entire table is certainly a very cumbersome thing to do especially when the table is very large or possibly infinite. So, when that is the situation you would like not to literally memorize the entire table, but to find means by which you

can calculate the output for a particular input by knowing or memorizing not all possible output input, input output pairs, but only a small number of input output pairs.

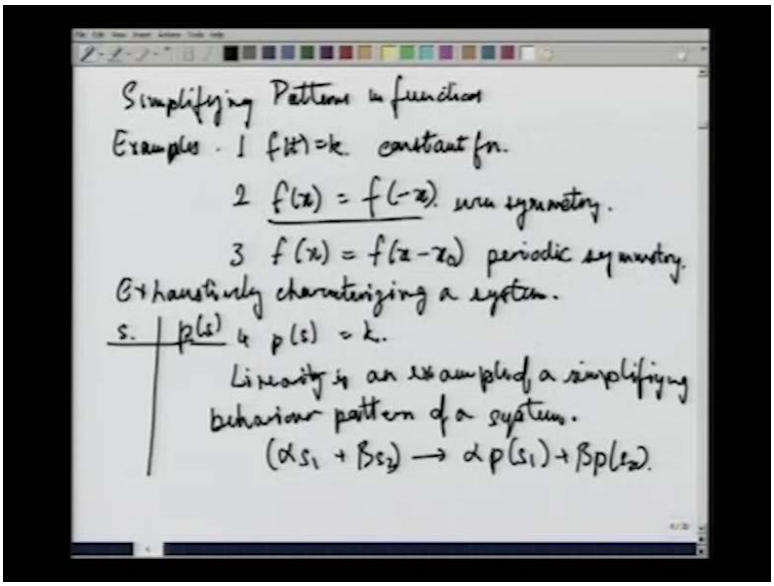
In short, you want to do the following simplification of the lookup table. Simplification of the virtual lookup table would result if you could detect patterns in the table. Any sort of regular pattern in the table, I used the pattern here in a very the word pattern over here in a very, very general sense a pattern can be anything that can be used to make a predicting calculation. For example, if I take the sequence of even numbers.

If I want to know what are all present what are all numbers are present in this set of even numbers one way to do. So, would be to literally memorize all possible even numbers which as you know is practically impossible because it is a infinite list. On the other hand I could have a different means of finding out if a number given to me is infinite or not. If I had had a complete list of course, I would have looked the number up in the list and if I found the number in the list I would have reported that it is an even number. If I did not find the number in the list I would declared that it is not an even number that is the brute force way of doing it.

What we want to do is and what you as you know we do is, if we want to find if a number is infinite or sorry, if a number is even we just try to divide the number by 2 and if it divides exactly without any reminder or without any fractional component then we will say that the number is even. So, we are replacing huge memory storage by an algorithm, in this case the algorithm of division by 2. And this algorithm is able to substitute for an infinitely log table and making and can making it finite table which merely tells you what to do that is divide by 2 and see if the result is an integer. If the result turns out to be an integer then the original number was even.

So, this is the sort of simplification that term is looking for. A simplification where some pattern can be detected using which an algorism can be designed or can be found which will give us the answers to the questions we have. Without having to go back to the brute force storage of all the function values this is what we want. Now, what kind of patterns could be there in a general function, for example,

(Refer Slide Time: 12:00)



Let us talk of simplifying patterns in functions. Examples; First example, v of t or if you like f of t equals k lets say. So, this is a function we want to see there is a pattern in this function there is very, very clearly a very, very dominating pattern here, which simply says that all rows in this lookup virtual lookup table are identical. So, you just have to know any one row and you know all the remaining rows. This is what you would call a constant function. Now suppose we try something else little more complicated.

Let us take an even function f of x equals f of minus x this is what we call an even valued function. Now does this help us in some way, what is the amount of freedom that this function has to express itself. In the first example that is f of x equal k it had absolutely no freedom at all. If you new x f of x for one particular value of x, then you knew f of x for all values of x, there was no room for variation there was no room for manipulation or change. Whereas now which the second example f of x equals f of minus x we have a slightly looser definition of a pattern.

So, lots of signals can satisfy this pattern. All we are saying is about x equal to 0 the function should say have the same value at a point if reflected across the 0 line the function has the same value at the other point as well. In short absolute value of x is an example of a function, which has a certain amount of symmetry a certain pattern in it. So, if you want to memorize such a function you can at least take the shortcut of memorizing one half of it and then declaring this feature of even symmetry.

If you say it is even symmetric and if you specify the function 0 to infinity that completely helps you to construct the rest of the function on the other side All right. So, let us look at more patterns in all cases as with the third pattern as well. Suppose we have a periodic function here f of x equals f of x minus x 0. Suppose this is what we called the periodic function or periodic symmetry. The whole idea is there a means by which you can have knowledge of only how the function is defined over a part of the domain. I am using this knowledge can you construct the entire function. Wherever and whenever you can do this you essentially have some symmetry or pattern in the function. And that is the kind of pattern that attracts our attention at the present time.

If you take a periodic symmetric function, you only need to know the value of function over one period. Now if you want to define the function over all time or over all x in this case. Then all you have to do is to construct a periodic extension of the part of the function that you are aware of that will give us the entire function. So, each of these things is an example of a symmetry. This is a very, very tight kind of symmetry pattern f of x equal to k or f of t equal k is a very, very strong symmetry, this is a very, very strong symmetry.

This is a relatively loose symmetry, because there are lots and lots functions which still can exits. Even if f of x equals f of minus x where as f of t equal to k there are that many different signals. That can have k that can distinguish f of t from each other if the all follow this pattern. This third example is somewhere in between first and second. It does allow certain amount of freedom of expression for the signal. At the same time it allows the user or the person who wants to follow the signal completely too just follows it for one period and then constructs it automatically for all time.

So, these are patterns, simplifying patterns that a signal that any function can have, and whenever you have a pattern then that helps you to condense the amount of information to be processed. Now, what is true for functions that are signals is also true for functions that are systems. So, suppose you have a system how do you normally characterize a system. How do you exhaustively characterize a system? If you want to exhaustively characterize a system, you would again have a lookup table a virtual lookup table. Where you would have the input signals say s on the left column and here you would say what is your output of the system which is say p of s this is the output signal.

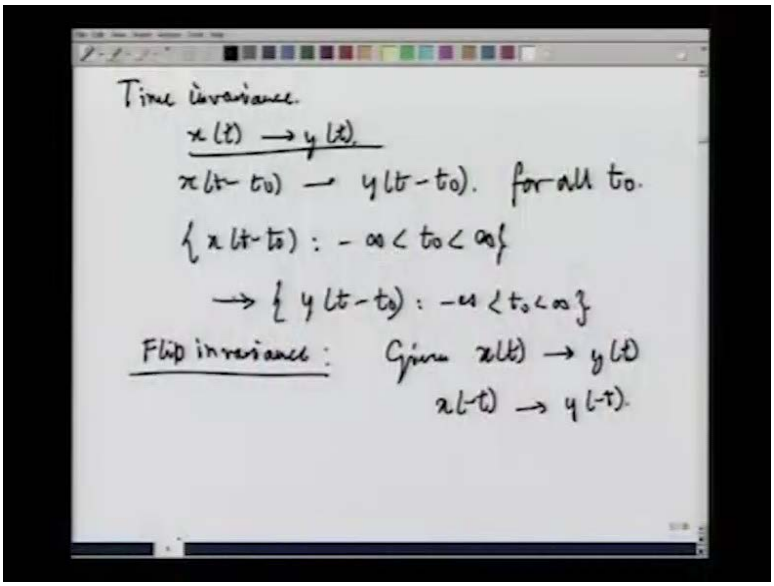
Now, this being a sufficiently typical example. Let us just see what kind of patterns of behavior, desirable patterns of behavior a processor can produce. One desirable pattern of behavior is to say that p of s is a constant, irrespective of the value of s over different in instance of time. So, you could say p of s. That has a very, very nice simplifying pattern. There could be other simplifying patterns that a system can have linearity itself is an example of a simplifying pattern of behavior for a system.

All right. Now how is it a simplifying pattern, because if I did not know that a system is linear or if the system is not linear? Then I would have to exhaustively list all possible input signals in the left column of the virtual lookup table for the system. And on the right side for each input I would have to make a record of what possible output will exist. On the other hand, if I just state the process the property of linearity. That helps us considerably reduce the size of the virtual lookup table that really needs to be carried in order to predict the behavior of the system. So, if you have a linear system, then if you recall the definition is this. This is the simplifying pattern, because suppose you just had one signal suppose you had x 1, now or s 1 as we are calling it suppose you had a signal s 1.

Now, in the lookup table s 1 appears in a certain place and against it is given the corresponding output of the system for s 1 being applied and that output is p of s 1 supposing you know this. Now, tomorrow suppose you wanted to apply two times s 1 to the same system, would it have been necessary to also record the corresponding response of two times of s 1 on the system is really not necessary, because the system is linear. If you know that s 1 yields p of s 1 then 2 s 1 will definably yield 2 p of s 1 as simple as that.

So, all cases of linear scale changes in s 1 will yield only similar liner scale changes in the output. And thus, we could replace a whole collection of different signals which are all scaled versions of s 1, by simply s 1 and a statement that the system is linear. All right. This makes a linear system already possessed of a certain great simplifying features.

(Refer Slide Time: 23:47)



Other examples of a simplifying behavioral pattern in a system come out of time invariance. What is time in variance, we had said that suppose you know that x t yields y t then, what would you get if you apply an x t minus t naught. You would simply get y t minus t naught and this would be true for all t naught. So, the entire set of rows in the virtual lookup table containing signals x t minus t naught, minus infinity less than t naught less than infinity will correspondingly map, because of time invariance to the change that are occurred over here, would be y t minus t naught, minus infinity less than. This is indeed a great simplification and it is these simplifying behavioral patterns that lie at the bottom of our seeking after a signal representation.

Now, what makes the business of simplifying behavioral patterns, what makes them interesting? One thing I said of course is that instead of handling the entire lookup table, virtual lookup table of the system. We only have to make a condensed table containing a few representative signals and their corresponding outputs you do not have to have all the shifted versions of x t. If you have written in the table somewhere that x t leads to y t, then you do not have to worry about x of t minus t naught for every different value of t naught you do not have to worry about this.

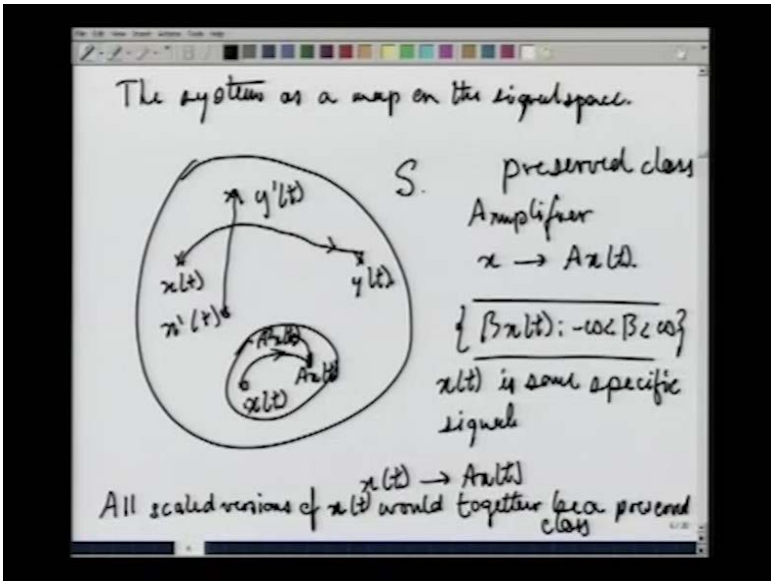
Similarly, with linearity if you knew that x t leads yields y t then you would know that k times x t will yield only k times y t. So, you do not have to say over and over again 2 x t will give you 2 y t, 21 x t will give you 21 y t and so on. You do not have to do it.

That is the beauty of having a symmetric behavior in the pattern of the system. Now let us see suppose you have certain family of processors or certain family of systems all of which are possessed of the same kind of symmetric behavior. What I am mean, to say is you do not have one time invariant system, but you consider the family of all time invariant systems or let us take a different case as a useful example instead of dealing with time invariance as a simplifying pattern of behavior.

Let us look at what I will call flip in-variance. I will say that a system is flip invariant, if given x t yields y t in the system in the output of the system, then x of minus t should simply yield y of minus t as well, fine. Now, suppose you have this property of a system given to you, then you see you do not really have to specify each signal for its entire length. If you say this is the value of x t for one half of time or rather if you collect the input output pairs for one set of x t then for all the time reversed signals corresponding to the members in the given set. You already know what value y t will take when x t is changed to x of minus t y t will change to y of minus t.

So, this is flip invariance and you can see that here again the table gets simplified. Now, what is our objective in using a representation, in seeking a representation? What I am going to say now is probably the most important part of this whole course. So, let us look at it very, very carefully. I will discuss this in a slightly unconventional manner not the way the text books go about it. I am going to discuss this by taking about the system as a map on the signal space, so the system as a map on the signal space.

(Refer Slide Time: 29:23)



Let us make a large Venn diagram of what we are looking it. Let us make a Venn diagram, let us say this is my signal space I will call it capital s. Now, we know that given any signal input signal say at this point. Since this is the entire signal space this given signal when applied to a particular system will yield an output signal which also lies in the entire signal space because all signals lie in the entire signal space.

So, let us say this is the output. Then you go about mapping this input signal to this output signal let me call this x t, let me call this y t. Fine. Now, suppose you had a different signal x dash t and this was the corresponding output let us say this is y dash t. So, then you would say x dash t goes to y dash t and you would just draw a line which connects this and this fine. So, for every input signal there is an output signal also in the same signal space. This would be exhaustive or brute force method of describing the behavior of the processor, behavior of the system. A simpler way would be to recognize that this system has certain behavioral patterns.

Thus what a behavioral pattern gives us is a certain kind of behavior in the mapping of the function. So, for example, suppose we say that there is a sub set of the signal space which the property that if you take a signal from which in the signal space. Then processor or the system that we are the concerned with for the time being will always take this signal to another signal within the same sub set. This kind of a sub set with this property that every signal, which every input signal, which lies inside the sub set will yield an output signal which also lies in the subset is what we will call a preserved class.

It can be argued that every favorable or desirable behavior pattern that we are trying to seek for the behavior of a system will imply the existence of one or other kind of preserved class. Let us take examples of systems where some kind of a behavioral pattern exits and see what the preserved classes come out to be. We begin this by just taking a simple amplifier. What as an amplifier do? It takes x t to some A times x t. So, this is what an amplifier does. So, what I do is. In fact, not just this particular amplifier, but if I take any amplifier at all which does just this thing.

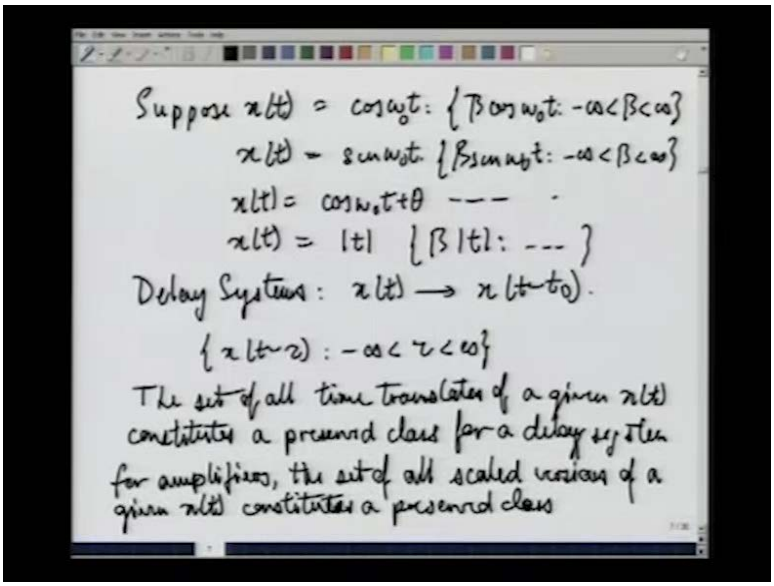
Then I am sure of this much. I am sure that whatever input signal is applied the form of the signal is preserved by the system and only its gain or its peak to peak value will be changed. So, what I do, I now want to find a preserved class, what kind of a preserved class? Let us see this is a preserved class. Let us take the set beta x t for minus infinity

less than beta less than infinity, where x t is some particular signal, x t is some specific signal. You will recognize that this preserved class or other this collection is indeed a preserved class under the operation of an amplifier a system that is an amplifier. For any amplifier system, this will constitute a preserved class. For different signals x 1 t, x 2 t, x 3 t you would have different preserved classes, beta x t, beta x 1 t, beta x 2 t for the all values of beta.

So, how many signals are there in a preserved class of this sort. An infinite number of signals because of beta can take an infinite number of different values. But yet, if I apply x t to the signal and I get A x t, after all A is also a number. So, for since beta x t since this class contains all beta x t for every value of beta it will also contains A x t.

So, if this for example, is x t then this might be A x t. And what if I took A x t itself as the input then, I would get A squared x t, where A squared is the new value of beta that is being invoked. So, once again it would lie within this set may it would go somewhere here and this could be A squared x t. So, all scaled versions of x t would constitute a preserved class under amplification. Would together constitute or be a preserved class under amplification. So, how many preserved classes do we have for the family of amplifiers, we have an infinite number of preserved classes.

(Refer Slide Time: 37:35)



One for each different x t that you can think of which differs by more than a scaled factor by something more than a scale factor from the other x t. Let us take example, suppose x

t equals cos omega t or cos omega naught t. Now, this is the x t, that I have the preserved class that contains this x t would be a cos sorry beta cos omega naught t going by the same notation. But now, suppose I took x t equals sin omega naught t. It is not found anywhere in this class it is an entirely different signal.

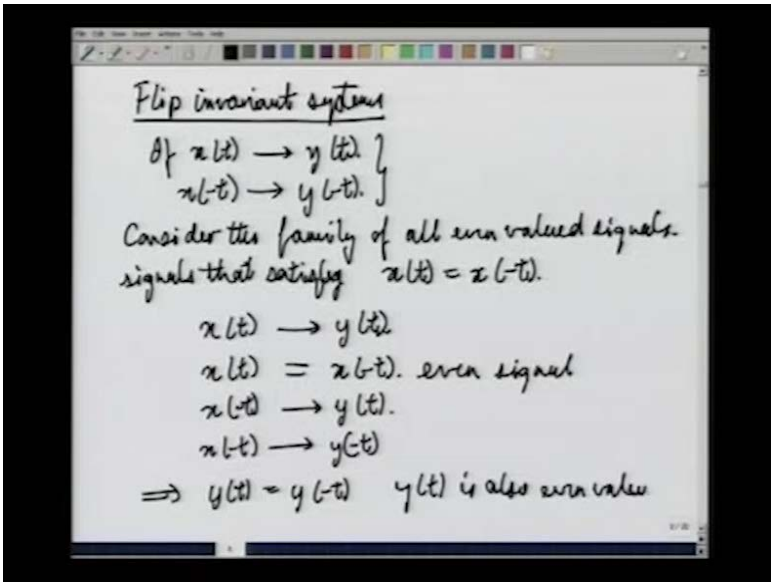
And hence it will give raise to its own class, beta sin omega naught t minus infinity less than beta less than infinity. Suppose instead I took x t equals cos omega naught t plus theta this again is not present in either of the above classes. It is al all together different signal. So, it has its own class consisting of all scaled versions of itself. Suppose you took x t equals mod t. This is another new signal all together new because it is not found in any of the previous preserved classes. So, it would give raise to its own preserved class consisting of beta mod t. So, each such signal not found in any other preserved class would give raise to its own preserved class and the nice thing about these preserved classes is the following.

Suppose I took an arbitrary signal and extra set has a sum of members of preserved classes. Then what the system would do to the sum could be stimulated by taking collectively into account what the system does to the members of preserved classes. More clarification on this in a little while, but let me now, for the time being proceed with more examples. This notion is so attractive and so beautiful. That it lies at the heart of the whole theory of signal representation and transforms. Let us take not amplifiers, but delay networks or delay systems characterized by x t yields x t minus t naught. Suppose every x t is shifted by a factor t naught and that is what the delay system does I am really not saying that it is a positive delay it could be a negative delay could be an advance.

Now, if we take any delay systems, what if any are it is preserved classes? Straight forward question, straightforward answer. Take x t, take this as an example; take the set of all x t minus tau for minus infinity less than tau less than infinity. This is a preserved class, why is it a preserved class take any x t in this class. What is the result of applying the signal to the system? The output will be x t minus t naught. If you apply x t minus tau as the input would be x t minus tau minus t naught, which is also already present somewhere else in this class. In short the set of all time translates of x t constitute a preserved class for a delay system. The set of all time translates of a given x t constitutes a preserved class for a delay system.

So, how many signals are there in a preserved class? An infinite number, because you can have an infinite number of different possible delays. How many preserved classes are there? Again there is an infinite number of preserved classes for a delay system, because you can have different signals x t each of which is not found in any other class. And for each such signal there is its own preserved class consisting of all time translates of this. Now, contrast this with over previous example, where we were dealing with amplifiers there what we had was the set of all scales of each x t. For amplifiers the set of all scaled versions of a given x t constitute a preserved class. Even more complex examples can be given.

(Refer Slide Time: 45:13)



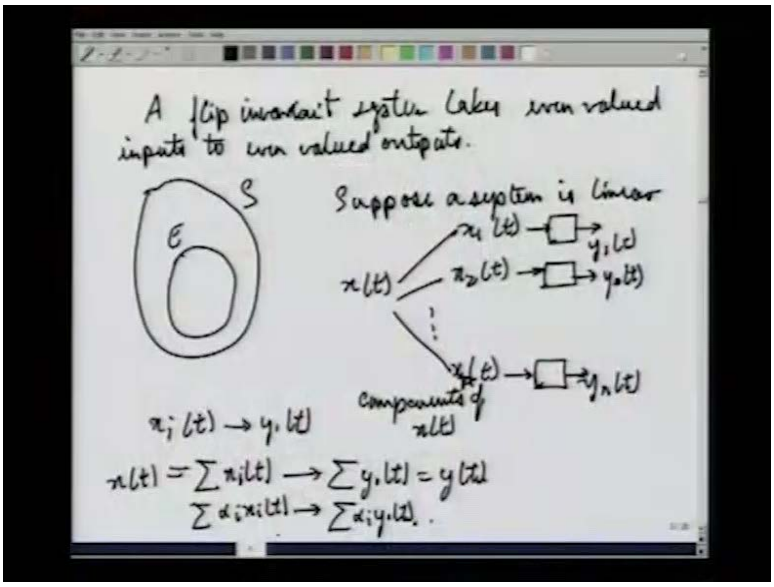
For example, suppose we consider the set of all flip invariant systems. What is a flip invariant system something we have will define yet? A flip invariant system is defined as follows. If the flip invariant system takes x t to y t then the same system should take x of minus t to y of minus t. We expect a system to be a flip invariant system if it follows this particular definition. This is a flip invariant system does this also have a preserved class yes it does. And the way to find that out is to look at the family of all even valued signals.

Consider the family of all even valued signals. That is signals, that satisfy x t equals x of minus t. Now this set of even valued signals can be shown to be preserved classes for the set of flip invariant systems. How do we show this? Take an even valued signal. And

apply it to a flip invariant system, it does not matter that we are not focusing on a particular flip invariant system. We are only concerned with any system which satisfies this particular property. The property of flip invariance in such system we will have the set of all even signals as a preserved class.

Now, take any x t, x t should give you some y t. We do not know whether if x t is even y t will be even or not as yet. But let us go ahead and see what happens. Suppose, x t gives you y t. Right, and suppose x t is even then x t equals x of minus t because x t equals x of minus t even valued signal. Then x of minus t is just another name for the same input x t and so it should also give you y t. On the other hand, we have the definition of flip invariance which says that if x of minus t is applied then the output will be y of minus t. Now, given these two things it is inescapable that y t is the same as the y of minus t which means that y t is also even value.

(Refer Slide Time: 49:23)



This means that a flip invariant system takes even valued inputs to even valued outputs, which is very nice because; that means, if you take inside the entire signal space s the set of all even valued signal which I will call e, then this family of signals is closed under processing may flip invariant systems. So, as you can see different kinds of behavioral patterns that might exists in the processor or in the system will yield certain kinds of preserved classes in the signal space. Of course, the which is the preserved class is

depend what preserved class is dependent upon what kind of a system we are concerned with, if the system changes its preserved classes will; obviously, also change.

So, different systems will have different preserved classes, but it often is also the case that a whole family of systems, which have a certain common behavioral pattern will all have the same set of preserved classes. For example, the set e the subset e of the signal space shown in this diagram over here, is a preserved class not for one particular flip invariant system, but to all flip invariant systems.

So, that is a nice thing to have. Now, where does this lead us? Where does this patterns of the existence, I mean this behavior of having preserved classes. Where does this really help us? What does it give us? In order to understand this question, we will have to go back to what I had said about understanding or modeling the behavior of a system. As I said, that one way of describing the entire system's behavior is to maintain a complete lookup table, a thorough lookup table which covers all possible inputs and all possible outputs.

But since, this is too cumbersome to handle, too cumbersome to process, too cumbersome to store. We are always in the search for shortcuts and the existence of preserved classes is one possible means of finding such shortcuts. Now, the shortcut that I am about to propose is generic to all kinds of transforms, and all transforms work on this belief that for a certain class of systems. They will exploit the existence of preserved classes in order to make a representation these means two things.

The business of signal representation is intimately tied up with the business of processing of those signals. If you want me to identify what kind of preserved classes I should take in interested. I should; obviously, start by asking what are the kinds of systems that you will be applying these signals to. Only when the answer to that question is known, will I able to identifying and isolate the appropriate preserved classes.

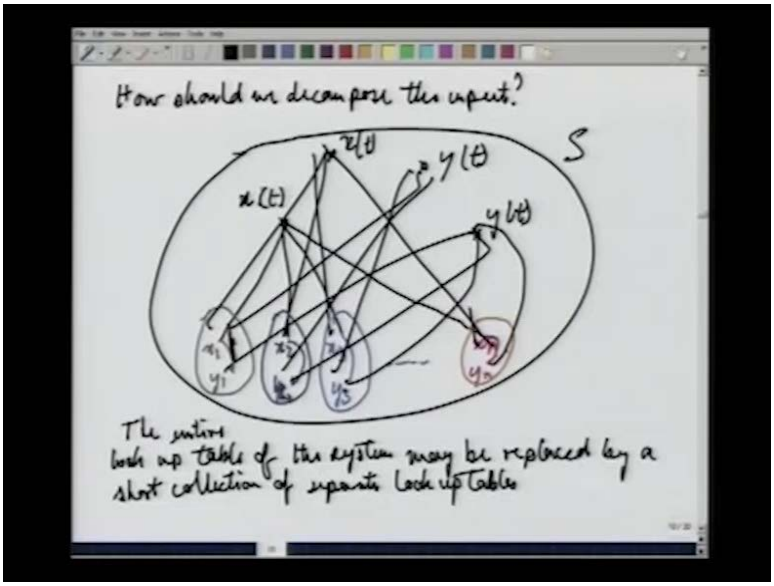
So, now suppose you have a set of preserved classes and suppose further that the system, let us say in the present context let us just take a simple assumption, let us assume that the systems is linear. Suppose, a system is linear; that means, it satisfies the theorem of superposition. If it satisfy the theorem of superposition, I could either apply a particular signal as the input and observe it is output or I could decompose the signal into a set of

components. Apply each component separately one at a time and note the responses and ultimately add all the responses.

I would then get a sum of responses whose value is identical to the sum of the components namely to the original input signal. Fine. This is possible whenever a system is linear. You decompose the signal in components, you have x t break keep it up into x 1 t, x 2 t and so on which are what I call components of x t. I still not made any, I have diverged anything about what kind of components, but we will soon be able to put two one two together about the earlier part of this lecture and the present part. And soon understand what we are looking for…

So, then when I have this I apply each component separately to the system. Let us say that x 1 t when applied to the system yields y 1 t, x 2 t yields y 2 t and x n t yields y n t you have all these different outputs. Because the system is linear, we know that given the fact that x I t yields y I t we know that x t which is equal to the summation of the x I t will give raise to summation of the y I t equals to y t.

(Refer Slide Time: 57:50)



Fine. Furthermore since the system is again linear. I can take different scaled versions of x I t and I would get the correspondingly scaled versions of the y I t and I could again add them. Say alpha I x I t would yield summation alpha I y I t correct. Now we have said all these, but we have not made any mention or made any comment upon what these components should be. Now, we see that if a system is linear and if by virtue of it is

linearity, it is homogeneity and it is additively. We can apply a set of components in parallel to an infinite number of copies, a large number of copies of the system and process each component output separately. Then we could recompose the component outputs to obtain the overall output to the overall input.

So, what kind of component inputs signals should we choose? How should we decompose the input signal? This is where the preserved classes come in. Let me draw a big diagram over here, a big Venn diagram. This my signal space. Now in this signal space I have different preserved classes this could be one preserved class, this could be another preserved class, this could be a third preserved class and so on, until say this is another preserved class.

Fine. Now, one way as I said to understand the behavior of the system to take any particular x t over here and find that it gets mapped to y t over here, but if you try to do that each x t would be a surprise because you would no which y t it would go to. What I do instead is this I first decompose x t taking one component from each preserved class that is available. So, this x 1 t will come from here, x 2 t will come from here, x 3 t will come from here, x n t lets say come from here. Though there is no need for n to be finite and just using it for demonstration sake.

Now, this is x 1, this is x 2, this is x 3 and this is x n. Now, what I am going to observe is this since these are all preserved classes of the system to which x t has been applied. We can be sure that x 1 will be taken to some other y 1 lying in its own class. x 2 will be taken to some other y 2 lying in its own class. x 3 will be taken to some y 3 also lying its own class and so on. For x n it will go to y n which lies in the same class. Fine. So, that is true for each of the components now that you have all the components of the output they would automatically yield a certain output upon combination. So, you would y 1 would go to this, y 2 would go to this, y 3 would go to this, y 4 would go to this. They would all add up and they would give you y 1 they would give you y t.

So, we seem to actually be going through a longer route. We take x t we decompose it into components. Process each component separately and combine all the output components to get the final output signal. So, what is the advantage in this the advantage in this is this. You really do not any longer need to memorize the entire virtual lookup

table of the system. You only need to memorize the lookup tables of the individual preserved classes.

So, the entire lookup table of the system may be replaced by a short collection of separate lookup tables, fine. With this we should be able to handle a large number of signals in the original signal space. Suppose instead I had something over here, this would decompose into something here, something here, something here and something here.

And if it did that it would give me the corresponding outputs which would also of course, be different from y t or from y 1 t to y n t. And their composition would yield the corresponding output y. So, you would get this going to somewhere here, this coming from here, this coming over from here and this coming over from here, so on contributing to the reconstruction of the output that would have occurred. If you are directly applied the original input without decomposition to the system.